UNIVERSITATEA POLITEHNICA BUCUREŞTI

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE

DEPARTAMENTUL CALCULATOARE

|  |  |
| --- | --- |
| upb | cs |

PROIECT DE DIPLOMĂ

Evidența concediilor într-o companie

|  |  |
| --- | --- |
| Coordonator ştiinţific:  Conf. Dr. Ing Alexandru Boicea | Absolvent:  Silaghi Alexandru-Mihail |

BUCUREŞTI

2016

Cuprins

[1. Introducere 1](#_Toc454747225)

[2. Nivelul de dezvoltare 2](#_Toc454747226)

[2.1 Modelul arhitectural MVC 4](#_Toc454747227)

[2.1.1 Modelul 4](#_Toc454747228)

[2.1.2 View 5](#_Toc454747229)

[2.1.3 Controllerul 5](#_Toc454747230)

[2.2 Aplicație web 5](#_Toc454747231)

[2.2.1 Aplicații web single-paged 6](#_Toc454747232)

[2.2.2 AJAX 7](#_Toc454747233)

[2.2.3 jQuery 8](#_Toc454747234)

[2.2.4 Bootstrap 8](#_Toc454747235)

[2.2.5 HTML5 9](#_Toc454747236)

[2.2.6 Thymeleaf 10](#_Toc454747237)

[2.3 Spring Framework 11](#_Toc454747238)

[2.3.1 Spring MVC 11](#_Toc454747239)

[2.3.2 Spring Boot 13](#_Toc454747240)

[2.3.3 Spring Security 14](#_Toc454747241)

[2.3.4 JSON Web Token 16](#_Toc454747242)

[2.3.5 Hibernate 16](#_Toc454747243)

[3. Proiecte similare 17](#_Toc454747244)

[3.1 Employee Vacation Tracking Calendar 18](#_Toc454747245)

[3.2 Timetastic 19](#_Toc454747246)

[3.3 WhosOFF 19](#_Toc454747247)

[4. Implementare 20](#_Toc454747248)

[4.1 Modelul de analiză al aplicației 21](#_Toc454747249)

[4.2 Detalii de implementare 23](#_Toc454747250)

[4.2.1 Securitatea aplicației 23](#_Toc454747251)

[4.2.2 Automatizarea procesului de administrare a cererilor 24](#_Toc454747252)

[4.2.3 Transmiterea de email 27](#_Toc454747253)

[4.2.4 Completarea documentului Word 27](#_Toc454747254)

[4.2.5 Atribuțiile administratorului 28](#_Toc454747255)

[4.2.6 Nivelul datelor 29](#_Toc454747256)

[4.2.7 Baza de date 29](#_Toc454747257)

[5. Performanța 31](#_Toc454747258)

[6. Concluzii 33](#_Toc454747259)

1. Introducere

În orice întreprindere, eficiența angajaților este unul dintre principalele elemente ce determină succesul ei. Fie că vorbim de o întreprindere ce construiește autocamioane fie de una ce produce software, sunt necesare mecanisme de observare și control al angajaților prin care managerii, administratorii sau șefii de echipă să îi dirijeze spre a fi cât mai eficienți.

Prin intermediul calculatorului, metodele de control au devenit automatizate, încărcătura administrării a trecut de la suport pe hârtie la afișarea și gestionarea prin intermediul aplicațiilor și persistarea în baze de date, completarea greoaie a zecilor de formulare a trecut din grija departamentului de resurse umane în grija aplicațiilor ce permit angajaților să își completeze formularele și administreze datele fără a mai fi necesară interacțiunea cu departamentul de HR.

Unul dintre cele mai ignorate procese administrative este administrarea cererilor și a zilelor de concediu. Consider că timpul liber al angajatului este la fel de importat ca și timpul pe care acesta îl petrece muncind în perspectiva de dezvoltare personală și creștere a eficienței. Majoritatea companiilor mici folosesc chiar și în zilele noastre calendare fizice unde își notează zilele în care vor lua concediu sau folosesc diverse metode de notare a zilelor de concediu prin intermediul Google Spreadsheets sau Microsoft Office Excel fără a face o persistare a datelor și o eventuală analiză a acestora.

Problema în momentul de față este generată de numărul redus de aplicații ce permit o mai bună gestionare a concediilor angajaților și posibilitatea de a analiza și eventual eficientiza acordarea acestora. Prin metodele vechi se pot pierde informații utile, folosirea lor poate duce la blocaje în care prea multi angajați iau concediu în același timp iar persoana responsabilă cu administrarea zilelor de concediu, de obicei managerul sau administratorul care are deja o multitudine de responsabilități, i se adaugă o responsabilitate suplimentară.

Cele câteva aplicații ce oferă funcționalitatea de persistare și centralizare a datelor și eficientizare a procesului de management au o serie de lipsuri pe care proiectul de față iși dorește să le acopere. Aceste aplicații au o modalitate de plată ce dezavantajeză firmele cu un număr mic și mediu de angajați în detrimentul marilor companii prin reducerea plății pe angajat. Companiile cu număr mare de angajați vor plăti mai puțin per angajat decât micile compani, acestea preferând să rămână la vechile metode de completare a zilelor de concediu pe calendar sau în tabele Excel. Clientul căreia îi este destinată aplicația are un număr redus de angajați prin urmare achiziționarea de software generic nu oferă avantaje ce se reflectă în prețul plătit. Alt deficit este generalitatea oferită de acestea, dintre soluțiile existente, pentru a face aplicația sub brandul companiei și integrarea ei cu sistemele companiei fiind necesare fonduri suplimentare, fonduri ce micile companii nu iși permit să le plătească. Generalitatea acestor aplicații se simte și în cadrul legislativ, neexistând suport pentru legislația de acordare a concediilor în România.

Proiectul de față își propune rezolvarea acestor probleme prin intermediul unei aplicații web ce oferă posibilitatea angjatului să se autoadministreze iar managerului să ofere posibilitatea de a observa tiparele comportamentale și a lua decizii de administrare pentru a balansa eventualele aglomerări de cereri de concediu pentru a păstra o eficiență a companiei sau a echipei constantă pe parcursul întregului an. De asemenea proiectul se va axa legile privind acordarea de zile libere pentru diferitele tipuri de concedii, definite în Codul Muncii. Pe lângă aceste cerințe clientul dorește adăugarea de diferite facilități suplimentare pentru alte dezvoltări ulterioare.

În proiect se întrevăd două tipuri de utilizatori, primul fiind utilizatorul comun ce depune cererile de concediu, interoghează baza de date pentru a afla numărul de zile de concediu rămase și administrează datele sale personale și administratorul , persoana ce adaugă utilizatori noi, aprobă cererile de concediu și interoghează baza de date pentru a afla detalii despre toți utilizatorii.

Fiind o aplicație web am preferat structura arhitecturală de tip Model View Controller ce permite dezvoltarea întregii aplicații intr-o singură pagină și diferențiază aplicația în trei module permițând refactorizarea într-un modul fără a impacta major structura aplicației. În această structură se regăsesc în interfață tehnologii precum HTML5, JavaScript, CSS pentru dezvoltarea la nivel vizual și legătura cu utilizatorul, legătura cu serverul și prin urmare cu Controllerul se face folosind AJAX și jQuery. Pe partea de server controllerul este concretizat folosind Spring MVC, Spring Boot și Spring Security iar comportamentul Modelului este definit folosind paradigma de programare orientată pe obiecte prin intermediul limbajului Java. Legătura cu nivelul datelor se face folosind Java Persistence API prin implementarea sa Hibernate, prin intermediul Spring Data iar datele sunt depozitate într-o bază de date MySQL.

În stadiul actual proiectul oferă posibilitatea utilizatorilor să depună cereri de concediu, să vadă numărul de zile rămase pentru fiecare tip de concediu în parte (concediu de odihna, medical și fără plată), să vadă și modifice diferitele date personale (număr de telefon, adresă de email, parolă) să obțină cererea efectivă sub format .docx și eventualele fișiere ce reprezintă trimiterile de la medicul de familie sau fișele de internare atașate cererilor de concediu medical. Administratorii au acces la datele tuturor angajaților, la cererile acestora de concediu și implicit asupra fișierelor atașate, pot aproba sau respinge cererile aflate în curs și pot vedea cu ajutorul unui calendar cererile de concediu în perioada următoare.

Lucrarea de față a fost structurată astfel încât să faciliteze dezvoltările ulterioare permițând dezvoltatorilor o înțelegere suficient de amplă a tehnologiilor folosite cât și a mecansimelor proprii de folosire pentru satisfacerea scopurilor proprii. În capitolul 2 este descris nivelul de dezvoltare actual, situarea în materie de tehnologii și arhitecturi punând totodată baza fundamentelor teoretice pentru înțelegerea funcționării aplicației și ulterior dezvoltării acesteia. Capitolul 3 expune detaliat proiectele similare ce au ca scop rezolvarea problemei procesului de automatizare a administrării cererilor de concediu, precizând atât puncte tari cât și puncte slabe și situațiile în care acestea sunt de preferat pentru a fi folosite. În prima parte a capitolului 4 este expusă pe larg motivația proiectului, detaliind cerințele clientului ce va folosi aplicația, argumentând arhitectura și tehnologiile folosite. În continuarea capitolului se drescriu modalitățile prin care aceste cerințe au fost rezolvate atat prin scenarii de business cât și prin implementarea concretă a acestora. Capitolul 5 prezintă o scurtă analiză a performanței aplicației iar în capitolul 6 se prezintă funcționalitățile viitoare și diverse concluzii.

1. Nivelul de dezvoltare

De la primi biți transmiși prin rețeaua dintre Universitatea din California și Institutul de Cercetare Stanford, la comunicarea dintre miliardele de utilizatori, Internetul a avut o dezvoltare fulgerătoare crescând exponențial. A facilitat numeroase activități, permițând omului modern un nou grad de confort iar în cazul companiilor facilități suplimentare pentru a spori eficiența angajaților. Folosind Internetul, magazinele fizice s-au transformat în magazine virtuale, industria jocurilor video s-a dezvoltat exploziv formând o industrie de 71 miliarde de dolari la nivel global în anul 2014 conform raportului Entertainment Software Association[[1]](#footnote-1) iar numărul de dezvoltatori și tehnologii pentru aplicațiile web au crescut exponențial. Trecerea de la Internetul comunicării între oameni la trecerea comunicării între dispozitive se dezvoltă accelerat ajungându-se treptat la conceptul de ”Internet of things”.

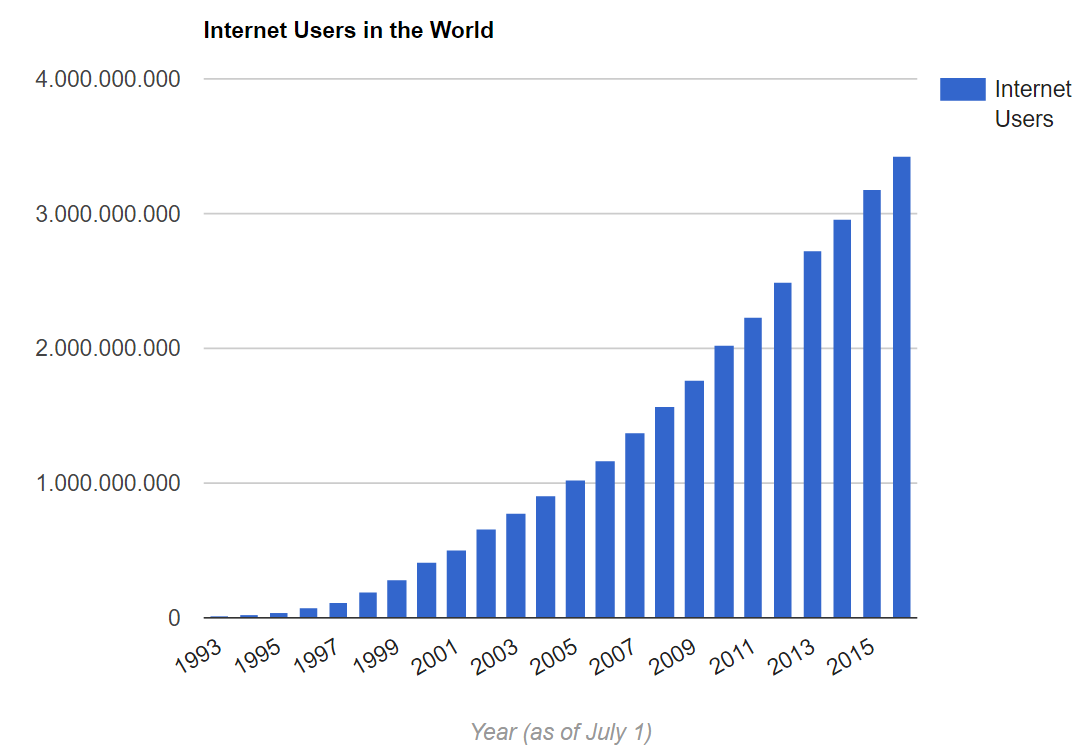
****

Fig 1. Numărul de utilizatori de Internet aproximat la 1 Iunie 2016[[2]](#footnote-2)

În domeniul afacerilor Internetul s-a dovedit o unealtă folositoare atât pentru dezvoltare(E-commerce) cât și pentru administrare, înlocuindu-se administrarea firmei cu ajutorul formularelor fizice de hârtie cu calculatoare legate la o rețea privată specifică firmei numită Intranet. Stimulat de întregul sector al afacerilor, Internetul a devenit un element esențial în administrarea și dezvoltarea oricărei afaceri.

Internetul a permis dezvoltarea a numeroase protocoale și servicii precum : File Transfer Protocol pentru transmiterea de fișiere, Simple Mail Transfer Protocol pentru transmiterea de e-mailuri, servicii de telefonie prin protocolul Voice over IP și multe altele. Printre cele mai cunoscute și utilizate servicii se află World Wide Web, serviciu format din 3 protocoale : Hyper Text Transfer Protocol, Hyper Text Markup Language și Uniform Resource Identifier au permis dezvoltarea și utilizarea paginilor web. Din 1989 când Sir Tim Berners-Lee a inventat acest serviciu până la momentul actual, forma și conținutul paginilor web s-au modificat radical. În forma sa inițială, WWW dorea partajarea de informații între utilizatori, comunicarea dintre aceștia fiind unidirecțională, emitentul oferind informații iar receptorul preluându-le fără a avea posibilitatea de feedback. Deși într-o formă rudimentară, rețeaua web oferea facilități neîntâlnite în alte servicii precum accesibilitatea, browserul având rolul de a afișa pagina web indiferent de sistemul de operare folosit, ușurința folosirii și aspectul global al paginilor. Deși avea numeroase beneficii, imposibilitatea receptorului de a da feedback a fost un dezavantaj major, obligând dezvoltarea de noi tehnologii pentru a-l rezolva. Noile tehnologii dezvoltate au stârnit interesul dezvoltatorilor de aplicații desktop ce iși doreau integrarea facilităților oferite de WWW în proiectele lor, oferind utilizatorilor posibilitatea de a rezolva taskuri prin intermediul interfețelor având ca platformă de producție site-uri web. Astfel iau naștere aplicațiile web, dezvoltate într-o singură pagină HTML, complexitatea și structura lor apropiindu-se mult de cea a aplicațiilor de tip desktop diferența fiind dată de modialitatea de prezentare prin intermediul browserului. [[3]](#footnote-3)

* 1. Modelul arhitectural MVC

Model View Controller este un model arhitectural ce permite separarea programului in trei secțiuni : Modelul ce reprezintă entitatea/entitățile cu care lucrează programul, View-ul ce reprezintă interfața grafică ce expune modelul către utilizator și Controllerul ce oferă posibilitatea utilizatorului sa modifice structura și comportamentul modelului.

În accepțiunea dezvoltatorilor de la Google[[4]](#endnote-1) , acest model este des întâlnit datorită evoluției limbajului JavaScript și popularizării aplicațiilor web ce folosesc cereri de tip AJAX. Cele trei elemente principale prezente în MVC sunt compatibile cu aceste tehnologii, controllerul incapsulând acțiunile utilizatorului surprinse de view prin intermediul cererilor asincrone, modelul ce primește o actualizare și trece intr-o nouă stare mai apoi notificând controllerul iar la final noua stare este expusă utilizatorului prin intermediul view-ului prin actualizarea acestuia.



Fig 1. Relațiile dintre entități în MVC1

* + 1. Modelul

Entitatea model încapsulează un obiect sau un ansamblu de obiecte ce descriu comportamentul aplicației. Acesta conține atât starea obiectelor cât și comportamentul lor, reprezentând abstractizat atât atributele cât și comportamentele obiectelor reale conținute modelate în aplicație. În dezvoltarea unei arhitecturi de tip MVC, modelul este deobicei primul element dezvoltat. Tot la acest nivel se implementează și logica de business a aplicației, logica de validare a datelor și persistarea obiectelor. Legatura cu baza de date se face prin intermediul unor obiecte speciale numite DAO (Data Access Object), facând o separare intre comportamentul obiectelor în logica de business și persistarea acestora.

* + 1. View

Entitatea vizuală reprezintă interfața între utilizator și program, preluând evenimentele generate de acesta și expunând noile stări ale modelelor obținute în urma evenimentelor. La acest nivel se dezvoltă partea grafică a aplicației, folosind tehnologii precum HTML, CSS, JavaScript, jQuery. Principalele obiective la acest nivel sunt prezentarea într-o manieră frumoasă din punct de vedere vizual și rapiditate în schimbarea conținutului în urma utilizării.

View-ul reprezintă cartea de vizită a aplicației, utilizatorul interacționând prima oară cu el. Dacă design-ul folosit nu reprezintă contextul în care funcționează aplicația, e supraîncărcat sau în general oferă greutate în a folosi aplicația, orice persoană va încerca să folosească aplicația va fi descurajată. La acest nivel se pune problema de User Experience, pentru o expreiență de utilizare bună este de preferat design-ul simplu, contrast între culori nu neapărat culori diferite. De asemenea așezarea elementelor în pagină este extrem de importantă. Utilizatorii folosesc diferite tipare în interacțiunea cu aplicația și au anumite așteptări ale unor elemente generice în pagina. De exemplu poziționarea logo-ului firmei/companiei/proiectului în colțul stânga sus sau a linkurilor de navigare în dreapta sus sau detaliile de contact în josul paginii.

* + 1. Controllerul

Elementul central în cadrul acestui model arhitectural, leagă modelele de partea de vizualizare, interpretează acțiunile din interfață și dirijează comportamentul modelelor, expunând noua stare utilizatorului. Dirijează fluxul informației de la view la model, separându-le pe cele două la nivel conceptual pentru a permite dezvoltarea de noi view-uri fără a modifica întreaga arhitectură, dândui acesteia flexibilitate la schimbare.

În esență controllerul este ceea ce ține legată întreaga aplicație, dirijorul întregului ansamblu. Un controller dezvoltat slab aduce cu sine dificultate suplimentară în refactorizare, făcând din aplicație un bloc compact, ”spagetti-code”, dezvoltarea ulterioară bazându-se pe înțelegerea funcționării intregii aplicații, în caz contrar modificările ulterioare producând și mai multă obscuritate.

* 1. Aplicație web

O aplicație web este "o aplicație ce utilizeaza rețeaua web sau tehnologii bazate pe browsere web pentru a rezolva unul sau mai multe task-uri peste o rețea, deobicei printr-un browser web" [[5]](#endnote-2). Acest termen folosit pentru prima oara in anul 1999 in documentația specificațiilor Java Servlet introduce noțiunea de interacțiune cu utilizatorul prin Web prin intermediul unui browser, fară ca acesta să instaleze un program specific aplicației folosite.

Diferența majoră între un site de prezentare și o aplicație web este felul în care acestea interacționează cu utilizatorul. Dacă intr-un site web interacțiunea bazeaza doar pe citire, o aplicație comunică cu utilizatorul, răspunde la acțiunile acestuia pentru atingerea unuia sau a mai multor scopuri. O altă diferență este autonomia aplicațiilor web, acestea neavând nevoie de programe adiționale pentru a funcționa, odată cu introducerea noului standard HTML5, programele adiționale (Add-ons) precum Adobe Flash Player au fost înlocuite, iar noi facilități precum geo-location și camera integration au permis o diversificare a temelor tratate de aplicațiile web, un exemplu in acest sens este noua extensie Facebook Live Stream ce foloseste aceste tehnologii pentru a permite utilizatorilor să faca live-stream fara a instala programe suplimentare. Aplicațiile web sunt interactive, răspund dinamic la evenimentele generate de utilizator, oferindu-i acestuia o experiență grafică placută, rapidă asemănatoare unei aplicații desktop.

Beneficiul major al acestui tip de aplicație este accesibilitatea, orice dispozitiv ce are aces la internet si poate susține un browser poate fi folosit ca instrument pentru utilizator. Din acest motiv ele sunt și foarte eficiente din punct de vedere al costului și al timpului de dezvoltare, pentru a face aplicația disponibilă pentru alte sisteme de operare sau altor dispozitive, modificările necesare sunt foarte puține, insesizabile comparat cu dezvoltarea altor tipuri de aplicații ce doresc diversificarea dispozitivelor sau sistemelor de operare. Un alt aspect pozitiv este trecerea calculelor intensiv computaționale de pe client pe server, aplicațiile web fiind o opțiune viabilă pentru aplicațiile dedicate dispozitivelor mobile ce nu au capacitate de procesare foarte mare. Există și un aspect negativ in acest sens datorită cerinței unei conexiuni stabile intre client și server, a costurilor legate de trafic și a lățimii de bandă. Problema funcționării aplicațiilor web in mod offline se dorește a fi rezolvată de Local Storage, Application Cache si a bazei de date oferite de HTML5, in modul offline aceste aplicații sa ofere totuși o funcționalitate minimală. În privința costului traficului și lățimii de bandă s-au dezvoltat diferite soluții precum împachetarea și minificarea bibliotecilor de CSS și JavaScript, reducerea datelor redundante din cadrul standardului HTML și aducerea fragmentelor de cod HTML de pe server în loc de a aduce toată pagina. Beneficiu suplimentar este lipsa necesității instalării unui program, un utilizatorii având acces imediat la funcționalitățile acesteia. Aplicațiile web în general sunt sigure și integre, neexistând probleme precum pirateria sau cracking-ul.

* + 1. Aplicații web single-paged

Paginile Web au avut o evoluție spectaculoasa incepând de la Web 1.0 când comunicarea intre utilizator si site era unidirecțională până in momentul de față la Web 3.0 când prin intermediul acestor tipuri de aplicații se poate efectua aproape orice tip de operație pe care ar putea să o facă o aplicație de tip desktop. Distanțarea dintre cele doua tipuri de site-uri web , definite in 1.0 si 3.0 a devenit și mai evidenta odată cu introducerea tehnologiei AJAX, astfel pentru a folosi un site nu a mai fost necesară deplasarea din pagină în pagină, actualizarea conținutului facându-se prin aducerea de fragmente de cod HTML la cerere, facând astfel posibilă crearea site-urilor single paged.

În accepțiunea aplicațiilor single paged (SPA)[[6]](#footnote-4) o pagină este ceea ce serverul trimite la primul contact cu utilizatorul. Această primă pagină este prima și singura pagină ce este complet analizată și reprezintă scheletul aplicației.Astfel prin acest model, logica interfeței utilizatorului este transferată de la server la client, serverul nu cunoaște starea in care se află aplicația, in funcție de interacțiunea utilizator-aplicație el oferă fragmente de cod HTML (templates), pe care dispozitivul client trebuie sa le interpreteze și sa le afișeze. Trebuie intărită disticția dintre pagină și view, deși la prima vedere SPA pare a schimba paginile, în realitate se schimbă doar view-urile, ferestrele văzute de utilizator, singura pagină complet interpretată de server fiind scheletul. Astfel in urma unei cereri asincrone a clientului serverul raspunde cu un set de date[[7]](#footnote-5). Acestea nu se limitează strict la un set de valori, de exemplu date scrise sub format JSON, conceptul fiind extins la tot ce este necesar clientului pentru a satisface o cerere, incepand de la date simple, pana la fragmente HTML , JavaScript sau CSS.

Printre beneficiile unui astfel de model se numară experiența fluidă oferită utilizatorului, aducând de la server doar fragmente de cod, aplicația web este mai rapidă, interacționează mai bine cu utilizatorul. Un alt beneficiu este reprezentat de cantitatea redusă de informații interschimbată cu serverul, traficul redus pentru dispozitivele mobile fiind un atuu.

Există de asemenea și o serie de dezavantaje, primul fiind dependența de JavaScript. Bazându-se in totalitate pe cereri asincrone și manipularea datelor pe partea de client, SPA are o dependență directă față de JavaScript și AJAX, facând utilizarea aplicației pentru clienții ce nu posedă aceste tehnologii imposibila. Un alt dezavantaj este reprezentat de imposibilitatea folosirii Search Engine Optimization datorită existenței unei singure pagini. Într-un SPA nu există noțiunea de pagini înlănțuite, astfel făcând inutile butoanele de back, forward si cancel in cadrul browserului. Deși in HTML5 o anumită stare poate fi pusă in stiva căutării, anumite beneficii precum intoarcerea la zona in care interacționai la apăsarea butonului back nu vor putea fi disponibile, ajungând să se refacă în JavaScript funcționalități deja existente in cadrul browserelor și care pot fi folosite in cadrul site-urilor multi-page. O altă problemă apare la indexarea și maparea paginii în cadrul algoritmilor de ranking, crawlerele (scripturile folosite de motoarele de căutare) nu interacționează cu cererile Ajax, prin urmare o astfel de aplicație este imposibil de clasificat și accesat[[8]](#footnote-6).

Comunitatea SPA a avut o explozie de tehnologii si practici in ultimii ani printre cele mai cunoscute tehnologii fiind bibliotecile de JavaScript : AngularJS, Meteor.js, Node.js, EmberJS ș.a ce ajută la dezvoltarea acestui tip de aplicație într-un timp cât mai scurt.

* + 1. AJAX

Ajax (Asynchronous JavaScript and XML) este un ansamblu de tehnologii ce permit aducerea datelor si actualizarea paginilor web chiar dupa ce acestea s-au incărcat. Termenul apare pentru prima dată in articolul lui James Garrett in anul 2005, acesta definind-ul astfel : "Ajax nu este o tehnologie. Este defapt un ansamblu de tehnologii, fiecare având rolul său, împreună aducând noi metode puternice. Ajax incorporează :

* Prezentare standardizată folosind XHTML și CSS
* Afișare dinamică și interacțiune folosind Document Object Model (DOM)
* Interschimbarea datelor si manipulare lor folosind XML și XSLT
* Obținerea de date asincron folosind XMLHttpRequest
* JavaScript ce incapsulează totul"[[9]](#endnote-3)

Ajax are la bază API-ul XMLHttpRequest ce permite comunicarea dintre client si server fără a reâncărca pagina. Numele API-ului nu mai are relevanță in aplicațiile moderne, acesta fiind extins de la date sub format XML la orice tip de date ajungând sa suporte chiar și alte protocoale înafară de HTTP precum suportă ftp și file transfer.

În cadrul acestui obiect, o serie de metode sunt mai des intalnite, metode ce configurează cererea, setează tipurile de date și raspund la evenimente.

Metoda .open(*metoda, url, [async=true]*) este metoda ce seteaza tipul cererii (GET, POST, DELETE, UPDATE etc.) apelată la adresa notata in câmpul URL. De observat faptul ca cererea Ajax poate fi sincronă deși această practică este de nedorit, fiind in detrimentul experienței utilizatorului.

Metoda .setRequestHeader(*name,val*) adaugă in header-ul cererii la cheia *name* valoarea *val*. Prin această metodă se setează tipul conținutului aplicației, cheia fiind "Content-Type" iar valoarea este un tip MIME (*Multipurpose Internet Mail Extensions).* Cele mai des întâlnite tipuri media sunt : "application/json" pentru transferul de date în format JSON, "application/x-www-form-urlencoded" pentru transferul de date din formulare HTML, "multipart/form-data" pentru transferul de date de dimensiuni mari, fișiere,fișiere video sau a textului cu caractere speciale și "text/html" pentru transferul de template-uri HTML.

Manipularea evenimentelor se face prin următorii event handlers : "onloadstart", "onprogress", "onabort", "onerror", "onload", "ontimeout"¸"onloadend" cărora li se pot asocia funcții JavaScript ce pot consuma evenimentul.

SPA au la bază acest ansamblu de tehnologii, prin această metodă efectuându-se interacțiunea client-server. Viteza de răspuns a aplicației se bazează pe capabilitatea Ajax de a aduce templateul cerut de utilizator și posibilitatea de a face aceste cereri asincron, prin cereri non-blocante.

* + 1. jQuery

Caracterizată în site-ul lor de prezentare, dezvoltatorii afirmă : "jQuery este o bibliotecă mică, rapidă si bogata in caractersitici ce facilitează parsarea si manipularea documentului HTML, manipularea evenimentelor si cererile Ajax folosind un API ușor de ințeles ce este compatibil cu o multitudine de browsere"[[10]](#endnote-4). jQuery se comporta ca wrapper peste JavaScript, facilitand selectarea elementelor din DOM prin functia $ ce intoarce un obiect JavaScript specific acestui framework. Functia $ implementeaza modelul fabrică (factory pattern), ce construieste un obiect conform selectorilor specificați in argument.Această bibliotecă a imprumutat o parte din selectorii oferiți de CSS și a adăugat câțiva noi. Cei mai deși întâlniți selectori sunt : selectorul după id, selectorul după atribut, selectorul după tip, selectorul după clasă, selectorul după copii.

Pentru a selecta obiectul cu id-ul *idObiect* și a-l instația ca obiect de tip jQuery este necesară construcția : $(“#idObiect”). Id-ul obiectului trebuie sa fie unic în cadrul DOM-ului, dacă există mai multe elemente cu același ID, comportamentul programului este nedefinit. Dacă dorim sa selectăm toate obiectele ce au un atribut cu o anumită valoare se folosește selectorul după atribut, acesta are urmatoarea formă : $(“[nume=’valoare’]”), acest selector intoarce un vector de obiecte ce au atributul nume egal cu valoare. Selectorul de clasă este reprezentat prin : $(“.numeClasă”) acesta intoarce o listă cu toate obiectele a căror clasă este specificată in ghilimele. Pentru a selecta toți descendenții direcți se poate folosi selectorul de copii : $(“părinte > copil”) ce va selecta toți descendenții direcți de tip copil ai căror element părinte este de tip părinte[[11]](#footnote-7).

Obiectele de tip jQuery au o multitudine de funcții ce permit manipularea DOM, schimbarea conținutului HTML, schimbarea stilului CSS elemente de legătură cu vecinii, părinții și copii, sau chiar distrugerea sau creearea de noi obiecte in interiorul DOM.

Un alt element important oferit de această bibliotecă este posibilitatea administrării evenimentelor din interfață. Catalogate in 5 categorii : evenimente in browser, evenimente la incărcarea documentului, evenimente în formular, evenimente alte tastaturii și evenimente ale mouse-ului permit dezvoltatorului ca într-o manieră rapidă și elegantă sa dea aplicației web posibilitatea de interacțiune cu utilizatorul, permit ca nici-o acțiune a acestuia in cadrul paginii să nu poată fi prinsă și consumată.

Printre principalele caracteristici oferite se află și încapsularea cererilor Ajax, ajutând la eliminerea de cod duplicat și construirea lor asemenea unui obiect de tip JavaScript

O caracteristică de bază a acestei biblioteci este posibilitatea de a creea plug-ins, cele mai importante fiind : jQuery UI pentru interacțiunea cu utilizatorul,diferite teme și widget-uri, jQuery Mobile pentru dispozitivele de tip mobil, Sizzle ca selector de CSS integrat in JavaScript și Qunit pentru unit tests in JavaScript. De asemenea dimensiunea redusă sub forma minificată, de numai 30kb conferă aplicației un timp de incărcare mic si un trafic redus facând din această bibliotecă una din cele mai facile pentru dezvoltarea de aplicații web pentru dispozitive mobile.

* + 1. Bootstrap

Bootstrap este un framework open-source dedicat front-end ce facilitează construirea unui site web prin eliminarea scrierii template-urilor CSS repetitive și ajuta la structurarea interfeței cu ajutorul sistemului de rânduri și coloane. Dimensiunea redusă a fișierului CSS și atenția sporită asupra responsivității prin adoptarea filozofiei mobile-first fac din acest frame-work o unealtă bună pentru o aplicație web cross-device. Mai adaugă alături de structură o serie de elemente de design precum design suplimentar pentru butoane, design pentru input, navbar, glyphicons ce dau aplicației un design modern, ușor de intreținut.

Un beneficiu adus de bootstrap este ușurința folosirii, sistemul de rânduri și coloane expunând o modalitate ușoară de a structura din punct de vedere vizual pagina.Structurarea sub formă de grid necesită un element ce are clasa ”container”. Acest element va fi părintele întregii structuri și va structura aliniamentul și distanța dintre elemententele ce se află în interiorul lui. Pentru a structura sub formă de rând este deajuns să definim un element ce are clasa ”row”. Dispunerea în cele 12 coloane se face folosind o convenție de nume : numele clasei va incepe cu ”col-” urmat de dimensiunea ecranului utilizatorului : ”xs” pentru dispozitive foarte mici, sub 768 de pixeli, ”sm” pentru dispozitive mici, tablete între 768 de pixeli si 992px, ”md” pentru dispozitive cu ecran mediu, desktop-uri intre 992px și 1200px, iar ”lg” pentru dispozitive cu ecran mare peste 1200px. După stabilirea dimensiunii ecranului dispozitivului folosit de utilizator se notează numărul de coloane ocupate din cele 12[[12]](#footnote-8).

* + 1. HTML5

Ultima standardizare în oferită de World Wide Web Consortium, HTML5 este încă la stadiul de W3C Working Draft, nu este complet, dar o parte din funcționalitățile oferite sunt deja implementate in majoritatea site-urilor si aplicațiilor web moderne. Incearcă să standardizeze modalitatea de dezvoltare a aplicațiilor web, structurând o suită de instrumente recomandate, puse in 3 grupuri : markup prin limbajul în sine HTML 5, presentation prin CSS3 și interaction prin DOM, Ajax și JavaScript.

Există numeroase beneficii aduse de noul standard printre acestea numărându-se acesibilitatea limbajului, codul HTML fiind mai ușor de citit datorită noilor taguri <nav>, <section>, <header>, <footer> și <article>.

Un alt beneficiu este simplitatea definirii atributelor DTD (Document Type Definition). Dacă in HTML4 care era bazat pe standardul SGML (Standard Generalized Markup Language) trebuiau declarate în doctype tipul de HTML folosit (Tranzactional, Strict sau Frameset), folosind o sintaxă dificilă și aducând un overhead semnificativ paginii, îngreunând procesul de dezvoltare, în HTML5 se folosește <!DOCTYPE html> ce încorporează DTD specificate anterior[[13]](#footnote-9).

Pentru aplicațiile multimedia, HTML 5 oferă un API special, facând mai ușoară afișarea și rularea acestui tip de fișiere fără a folosi o aplicație de tip ”third-party” precum Adobe Flash Player. Oferă posibilitatea construirii de elemente de tip <video> sau <audio> ce simplifcă efortul dezvoltatorului, fiecare browser având integrate modalități de a reda conținutul prin buton de play, de a reda si de a opri rularea fișierului media. O problemă în acest este datorată inconsistenței pe piață a codec-urilor folosite. Dacă inițial W3C a dorit impunerea a două codec-uri : ”Ogg Vorbis” pentru fișierele de tip audio și ”Ogg Theora” pentru cele de tip video, în urma obiecțiilor Apple și Nokia, W3C a retras standardizarea codec-urilor ramânând ca dezvoltatorii de browsere să decidă programele folosite. În momentul de față codecul folosit în majoritatea browserelor web este VP9, oferit de compania Google, un codec gratuit, fară drepturi de autor, integrat momentan in browserele : Chromium, Chrome, Firefox și Opera[[14]](#footnote-10).

Noul element <canvas> a permis aplicațiilor web posibilitatea randării și modelării imaginilor bidimensionale cât și a programării acestora folosind JavaScript. Acest element determină o zonă în pagina web în care programatorul poate să dezvolte aplicații grafice sau care să interacționeze cu utilizatorul. Acesta deschide posibilitatea extinderii aplicațiilor web și în industria jocurilor oferind posibilitatea dezvoltării acestora doar cu HTML si JavaScript.

Altă caracteristică HTML5 este geolocalizarea, dând posiblitatea dezvoltatorului dacă are permisiune de la utilizator sa afle poziționarea acestuia. Prin acest API se elimină dependența de entităților third-party, dând aplicațiilor web o implementare cât mai compactă. Poziția utilizatorului se poate afla folosind funcția getCurrentPosition() ce intoarce coordonatele carteziene ale dispozitivului folosit de utilizator, ce cu ajutorul unui serviciu de hărți poate fi afișată și folosită. Geolocalizarea poate fi folosită la potențialul său maxim in aplicațiile web pentru dispozitive mobile, încorporate cu GPS, permițând acestora să folosească pe deplin facilitățile oferite de dispozitiv in mediul web fără aplicații adiționale.

Inițial în specificațiile W3C pentru HTML5 se dorea să se introducă o bază de date ce folosea SQL numit WebSQL dar acest proiect a fost oprit. Motivul specificat pe site-ul W3C este : ”Specificarea a ajuns într-un impas : toți implementatorii intersați au folost același SQL in partea de backend (Sqlite), dar nouă ne trebuiesc multiple implementări diferite pentru a putea continua pe drumul standardizării”[[15]](#footnote-11). În locul acestei modalități de păstrare a datelor pe dispozitivul uitilizatorului s-au dezvoltat alte două proiecte : Indexed Database API și Local Storage. Descrisă pe site-ul w3.org drept o bază de date ce persistă datele sub formă de tuplu cheie-valoare/valori, datele fiind structurate sub forma unui arbore de tip B fiind considerat eficient atât pentru inserare și ștergere cât și pentru citire dintr-un set mare de date. Accesul la date este rapid, toate cheile fiind indexate. Local Storage este dorit să înlocuiască cookie-urile, acestea având o serie de deazvantaje, permițând doar 4kb de date, datorită expunerii acestora în text au fost folosite pentru a accesa malițios datele utilizatorilor și persistând doar in fereastra curentă. Accesarea datelor folosind Local Storage este facilă, ele fiind persistate sub forma perechii cheie-valoare, atât cheia cât și valoarea fiind șiruri de caractere. Persistarea datelor se face folosind obiectul global de JavaScript localStorage folosind metoda setItem(*cheie,valoare*). Pentru a obține obiectele se foloseste metoda getItem(*cheie*). Introducerea obiectelor de JavaScript mai complexe in acest fel nu este posibilă, de aceea ce-a mai intalnită metodă este transformarea acestuia in șir de caractere de forma JSON și apoi stocat.

* + 1. Thymeleaf

Thymeleaf este un template engine specific limbajului Java ce permite generarea de pagini HTML dinamice. Respectă standardele HTML5 și XHTML fiind folosit ca tehnologie principală in nivelul de prezentare in arhitecturi de tip MVC. Principalul țel al acestei tehnologii este : ”de a aduce template-urile naturale in procesul de dezvoltare, codul HTML ce poate fi corect afișat in browser să funcționeze și în prototipuri statice, permițând o colaborare mai strânsă între echipele de dezvoltare”[[16]](#endnote-5). Template-urile thymeleaf sunt mai ușor de citit și înțeles datorită păstrării formei de tip HTML spre deosebire de JSP necesită schimbarea tag-urilor pentru a genera cod. Vine ca un wrapper peste JSP, pentru a genera pagini dinamice având în continuare nevoie de obiecte de tip Bean trimise ca parametrii. Un alt beneficiu este reprezentat de posibilitatea rapidă de actualizare pe server, dacă la JSP este necesară reactualizarea intregului modul pentru interpretarea template-ului, thymeleaf suporta ”hot-deploy” ce permite actualizarea paginilor fără a mai fi necesar acest pas, facând dezvoltarea interfeței mult mai rapidă.

* 1. Spring Framework

Spring este o aplicație de tip Open-Source ce oferă dezvoltatorilor psobilitatea de a dezvolta aplicații pentru întreprinderi într-o manieră structurată, mai rapid, scriind cât mai puțin cod. Abordează toate aspectele unei aplicații, conținând elemente pentru toate nivelele de dezvoltare. La baza abordării Spring stă inversarea controlului (Inversion of Control – IoC) și mai precis injectarea dependențelor (Dependency Injection) acesta facând aplicațiile dezvoltate în Spring să fie slab legate (losely connected) păstrând totuși o coeziune bună în interiorul modulelor.

Conform principiului de inversare a controlului, în loc ca aplicația să controleze fluxul datelor, acesta este predat succesiv fiecărui modul, asemănător expresiei ”Nu mă suna tu, te voi suna eu”, framework-ul aplează codul aplicației, coordonând fluxul informației. Pe baza acestui principiu Martin Fowler a definit un nou ”flavor” de inversare a controlului numit ”Dependency Injection” și l-a definit astfel : ”Injectarea dependențelor se bazează mai mult pe construcții specifice limbajului Java decât pe folosirea interfețelor specifice framework-urilor. În loc de a folosi API-ul oferit de framework pentru a-și rezolva dependențele cum ar fi parametrii de configurare și obiectele colaborative, de către codul aplicației, clasele aplicației iși expun dependențele folosind metode sau constructori astfel încât framework-ul să decidă valorile corecte la runtime, bazate pe configurare.”[[17]](#footnote-12)

Beneficiile acestei abordări sunt :

* Explicitatea dependențelor – Codul auto-documnetează clasa, expunând in mod direct clasele de care depinde.
* Elasticitatea la schimbare – În cazul schimbării sursei de configurare, codul aplicației rămâne constant, fiecare dependență acționând asemenea unei interfețe, Spring legând interfața de o implementare ce are comportamentul dorit.
* Separarea aspectelor – Există o linie bine definită în codul ce exprimă comportamentul de business de cel de management al configurărilor, permițând astfel o coeziune puternică menținând în același timp o legătură slabă.
  + 1. Spring MVC

Legătura dintre aplicațiile web și Spring se face prin framework-ul SpringMVC. La baza acestui framework se afla Dispatcher Servlet, un servlet fiind un obiect Java specializat în tratarea cererilor HTTP. Acesta dirijează cererile către controllerele ce le pot consuma cererile, permițând deasemenea și un control mai aprofundat asupra datelor trimise. Pentru răspunsul cererii HTTP, servletul preia view-ul definit in controller și il intoarce pentru a fi expus utilizatorului. Configurarea Dispatcher Servlet se poate face în doua feluri : prin intermediul unui fișier de mapare : web.xml unde se specifică maparea fiecărui URI pe controller sau folosind Spring Boot și adnotarea @EnableWebMvc ce activează o scanare a componentelor pentru a descoperi clasele controller și a le mapa in mod automat. ”Spring suportă o multitudine de configurări ale mapărilor, in funcție de rolul utilizatorului logat, expunerea aplicației chiar și a perioadei zilei”6.



Fig. 3 DispatcherServlet ca interfață pentru HTTP Requests/Responses și Controllers/Views[[18]](#footnote-13)

Controllerele sunt clase specializate, ce au adnotarea @Controller în interiorul cărora există una sau mai multe metode ce au adnotarea @RequestMapping. Dacă este pusă adnotarea @EnableMVC aplicației, Spring va scana pachetul primit ca parametru JNDI, găsind clasele ce au adnotarea @Controller, preluând fiecare metodă și mapând-o la URI-ul definit in argumentele adnotării, astfel obținându-se un fișier web.xml automat, fără a mai fi necesară vre-o configurare din partea dezvoltatorului. O altă atribuție a controllerului este aceea de a mapa datele primite din interfață prin cererea HTTP la obiectul model, cu condiția ca acesta sa fie de tip Java Bean (pentru atributele obiectului să existe metode de tip Getter/Setter). Spring oferă posibilitatea controlului mapării la runtime cu ajutorul adnotărilor, astfel o mapare a unui URL se poate face la nivel de tip de cerere (GET, POST, PUT etc.), dacă are parametru de cale, în funcție de rolul utilizatorului logat și reținut in contextul aplicației.[[19]](#endnote-6) Tipul cererii determină deasemenea comportamentul controllerelor, prin adnotări prin parametrul ”consumes” și ”produces”, metoda definind atât forma datelor primite din interfață cât și tipul datelor intoarse în interfață. Manipulează deasemenea și forma view-ului , putând imbogăți forma acestuia cu obiecte din cadrul contextului aplicației. Pentru a pasa obiecte view-ului este necesară o instanță a unui obiect de tip Model ce ce va fi pasat mai departe către view 9. Obiectul de tip model odată declarat in lista de parametrii a metodei i se pot adăuga obiecte folosind metoda .add(*numeObiectÎnView, obiect*), obiectul putând fi accesat după numele declarat in metodă.

Metodele ce au adnotarea @ModelAttribute pot adăuga un singur atribut în mod implicit, sau mai multe folosind un obiect de tip Model primit ca parametru. În structura unui controller, înainte de a apela metoda ce mapează cererea, se aplează toate metodele ce au adnotarea @ModelAttribute, construind astfel toate atributele comune ale modelului folosit de controller evitând astfel crearea de cod duplicat. Pentru extragerea atributelor din cereri se poate folosi aceeși adnotare dar pe argumentele metodelor de mapare. Spring când găsește un obiect cu această adnotare va incerca sa-l extragă din model, în cazul în care acesta nu există, îl va instanția și il va depune în model. Atributele obiectului vor fi populate pe parcurs folosind cererile HTTP, preluând variabilele ale căror chei sunt compatibile, folosind metodele setter atașate obiectului 9.

Transmiterea de fișiere în Spring se face cu ajutorul unei clase speciale : MultipartFile. Cererile ce pot fi mapate pe acest tip de fişier trebuie să consume tipul media ”multipart/form-data”. RFC 1867 definește ”multipart/formdata” astfel ” Tipul media multipart/form-data urmează regulile tuturor stream-urilor de date de tip MIME multipart definite în RFC 1521. Este intenționat a fi folosit pentru a întoarce datele urmate completării unui formular... Multipart/form-data conține o serie de părți. Fiecare parte conține un header de descriere a conținutului unde valoarea este ”form-data” și numele atributului specifică numele câmpului din interiorul formularului..Asemenea tuturor tipurilor MIME multipart, fiecare parte are un câmp opțional Content-Type, cu valoarea implicită ”text/plain”. Dacă conținutul fișierului este obținut în urma completării unui formular atunci tipul părții este indentificat ca ”application/octet-stream” sau dacă este cunoscut, tipu media corect. Dacă mai multe fișiere sunt întoarse în același formular, tipul devine multipart/mixed, încorporat în interiorul ”multipart/form-data”[[20]](#footnote-14). Obiectul MultipartFile este considerat in API-ul Spring drept : ”o reprezentare a unui fișier încărcat primit într-o cerere de tip multipart. Fișierele sunt fie stocate în memorie sau temporar pe disk, indiferent de caz, utilizatorul este responsabil de a copia și persista fișierul așa cum își dorește. La fârșitul procesării cererii, toate fișierele temporare vor fi șterse”[[21]](#footnote-15) . Are atașate o serie de metode ce permit manipulearea fișierelor primite în interfață precum obținerea biților, a tipului media și copierea acestuia într-un fișier dat. Configurarea fișierelor de tip multipart se poate face în două modalități, fie prin definirea lor în fișierul application.properties, fie prin obiectul MultipartConfigElement cu adnotarea @Bean. Două dintre setările oferite sunt : multipart.maxFileSize ce determină dimensiunea maximă acceptată a fișierelor, iar a doua multipart.maxRequestsSize ce determină dimensiunea maximă a unei cereri de tip multipart/form-data.[[22]](#footnote-16)

După consumarea cererii HTTP, controllerul oferă un View, acesta poate să fie explicit prin intermediul unor valori de tip String, View sau ModelAndView sau implicit prin intermediul unor convenții[[23]](#footnote-17). Comportamentul răspunsului poate fi definit fie în interiorul controllerului, dar acestă metodă ar lega prea tare controllerul de răspuns, fie prin folosirea unor prefixe in interiorul numelui view-ului întors în cazul în care valoarea este de tip String. Prefixul ”redirect : cale” redirecționează utilizatorul către calea dată ca parametru. Alt prefix ”forward : cale”, predă modelul către calea din parametru, făcând astfel posibilă înlănțuirea de mai multe view-resolvers. Thymeleaf fiind integrat cu Spring MVC, aduce un prefix specific, ”numeView :: numeFragment”, oferind posibilitatea ca, din intregul view să se întoarcă doar un fragment, fragment marcat cu th:fragment=”numeFragment”.

* + 1. Spring Boot

Conform documentației oferite de Spring,”Spring boot ajută la crearea de aplicații de sine stătătoare, de nivel de producție bazate pe Spring ce funcționează direct. Majoritatea aplicațiilor de tip Spring Boot au nevoie de foarte puțină configurare. Scopurile Spring Boot sunt :

* Oferirea unei experiențe de dezvoltare mult mai rapidă și mai accesibilă pentru toate tehnologiile Spring.
* Oferă o serie de caracteristici specifice proiectelor de dimensiuni mari, servere încorporate, securitate, metrici, validarea datelor și configurare externalizată.
* Lipsa configurațiilor de tip XML şi a codului auto-generat”[[24]](#footnote-18)

Spring Boot işi propune să inlocuiască efortul de configurare a dependenţelor aplicaţiei, acesta facându-se automat în funcţie de dependeţele declarate in POM-ul Maven. De exemplu dacă declari in ca dependenţă JPA şi un driver MYSQL, Spring Boot va face configurările de bază.[[25]](#footnote-19)

Eficiența și flexibilitatea Spring Boot se observă și prin ”împachetarea aplicației și folosirea unui server HTTP încorporat permițând folosirea aplicației oriunde”[[26]](#endnote-7) permițând aplicațiilor web ce folosesc acest framework existența unei metode de tip main, apropiindu-le de aplicațiile desktop. Cum aplicațiile Spring Boot sunt în esență aplicații Java, ”Hot-swapping oferit de Java Virtual Machine ar trebui să funcționeze în mod automat, totuși pentru soluții complete se pot folosi JRebel sau Spring Loaded”[[27]](#footnote-20)

Printre facilitățile oferite de Spring Boot, se află clasa SpringApplication ”ce poate fi pornită din metoda main() unde este deajuns să se delege metoda statică .run()”7. Pentru aplicațiile web, această clasă va incerca sa creeze ”tipul potrivit de context”[[28]](#footnote-21), trecând obligația de dezvoltare a contextului de la utilizator la framework.

Configurarea aplicației se face cu ajutorul unui fișier ”application.properties”, Spring va căuta în mod automat acest fișier și ”îl va integra in mediul Spring : în subdirectorul /config al directorului curent, în directorul curent, în pachete de tip /config sau în rădăcină”7

Pentru logare Spring Boot folosește commons logging tot odată permițând existența altor sisteme de logare, precum log4j oferit de cei de la Apache, acesta necesitând un fișier de configurare suplimentar numit log4j.properties unde se poate seta nivelul de logare (DEBUG, ERROR, WARN) pentru un anumit pachet, sau pentru întreaga aplicație, se poate seta fișierul în care se va loga și multe altele.

* + 1. Spring Security

Spring Security este framework-ul oferit de Spring pentru partea de securitate, alături de celelate framework-uri oferind o soluție completă pentru dezvoltarea aplicațiilor de tip ”enterprise”. Dezvoltatorii Spring afirmă în documentația framework-ului că ”utilizatorii folosesc Spring Security din multe motive dar majoritatea sunt atrași după ce descoperă că elementele de securitate oferite de J2EE și EJB nu au aceeași adâncime, necesare în dezvoltarea aplicațiilor enterprise”7

Framework-ul se axează ”pe cele două zone majore ale securității : autentificarea și controlul accesului... Autentificarea este procesul de a stabili dacă utilizatorul are identitatea pe care o afirmă și controlul accesului în care se decide dacă un utilizator poate acționa în interiorul aplicației.”[[29]](#endnote-8) Cel mai important element în cadrul aplicației este SecurityContextHolder, descris în documentație : ”SecurityContextHolder este locul unde se păstrează toate detaliile legate de contextul curent al aplicației, inclusiv detalii legate de utilizatorul curent logat”[[30]](#endnote-9). La logarea utilizatorului, prin AuthenticationPoint se poate crea o instanță a obiectului Authentication unde se persistă detalii despre utilizator prin obiectul UserDetails. Obiectul Authentification astfel obținut este depus in ApplicationContext și păstrat până la log-out sau o anumită perioadă de timp până la expirare. Astfel se rezolvă prima parte a securității, autentificarea. Pentru partea de control al acesului, utilizatorului prin obiectul UserDetails i se pot asocia diferite autorități sau roluri, în funcție de acestea având posibilitatea de a accesa diferite resurse sau comportamente.

Pentru asigurarea securității, Spring Security folosește o înlănțuire de filtre fiecare având rolul său. Ordinea acestora este importantă datorită dependențelor dintre ele, așadar ”indiferent de filtrele specializate adăugate de utilizatori, ordinea filtrelor ar trebui să fie astfel :

* ChannelProcessingFilter, pentru redirectarea către un protocol diferit
* SecurityContextPersistenceFilter pentru stabilirea unui SecurityContext ce va fi pus în SecurityContextHolder la începutul fiecăriei cereri
* ConcurrentSessionFilter, dependent de SecurityContextPersistenceFilter prin SecurityContextHolder, pentru cererile următoare de la utilizator.
* Mecanisme de autentificare (UsernamePasswordAuthenticationFilter, etc.) pentru a putea modifica SecurityContextHolder astfel încât să conțina un obiect de tip Authentication valid.
* SecurityContextHolderAwareRequestFilter în cazul în care se folosește Spring Security ce foloseste HttpServletRequestWrapper.
* RememberMeAuthenticationFilter pentru a păstra obiectul Authentication o perioadă de timp
* AnonymusAuthenticationFilter ca în cazul în care nu s-a făcut nici-o autentificare să se adauge un obiect Authentication anonim.
* ExceptionTranslationFilter pentru interceptarea excepțiilor Spring Security astfel încât să se întoarcă un mesaj relevant în caz de eroare sau să se întoarcă un AuthenticationEntryPoint în caz de succes.
* FilterSecurityInterceptor pentru a proteja URI-ul aplicației și a ridica excepții de acces nepermis.”[[31]](#footnote-22)

Accesul utilizatorilor la resurse se poate face prin două metode, cu ajutorul adnotărilor sau folosind fișierul web.xml, unde se face rutarea URL-urilor aplicației. Pe baza unor expresii se stabilește validitatea accesului la resurse a utilizatorului astfel : hasRole(ROL) permite accesul la resurse doar dacă în obiectul UserDetails în atributul Authentication există o instanță a obiectului având numele ”ROLE\_ROL”. Expresia hasAnyRole(Rol1, Rol2) permite unei mulțimi de roluri să acceseze resursa. Cu ajutorul expresiei isAuthenticated() se restricționează accesul doar utilizatorilor ce au o instanță a obiectului Authentication depusă in SecurityContextHolder ce conține credențialele acestuia. Restricționarea prin adnotare se face facil, exemplu @hasRole(”Admin”) va permite accesul la resursă doar utilizatorilor ce au rolul ”Admin”. Pentru restricționarea prin xml, în fișierul web.xml se va adăuga :

<http use-expressions="true">

<intercept-url pattern="/admin\*" access="hasRole('admin')"/>

</http>

Fig. 4 Adăugarea restricțiilor în web.xml[[32]](#footnote-23)

* + 1. JSON Web Token

JSON Web Token (JWT) este un standard folosit pentru a transmite datele în siguranță între două entități. Definit în RFC 7519, JWT este ”o modalitate compactă și sigură de a reprezenta afirmații ce vor fi transmise între două entități. Afirmațiile în JWT sunt structurate sub forma un orbiecte de tip JSON ce sunt folosite ca încărcătură pentru o structura JSON Web Signature...permițând afirmațiilor să fie semnate digital sau protejate printr-un Message Authentication Code și criptate”[[33]](#endnote-10) . Cu alte cuvinte JWT permite păstrarea și transmiterea datelor de autentificare ale utilizatorilor într-un mod sigur, făcându-l un candidat bun pentru autentificarea în aplicațiile web. Un JWT este compus din trei părți : header, încărcătură și semnătura, criptările acestora fiind delimitate prin ”.”. În partea de header se definește tipul tokenului și algoritmul de criptare, informând destinatarul de modalitatea de decriptare. Încărcătura este compusă din afirmații, fiecare dintre acestea fiind puse sub forma JSON, structurate în perechi cheie valoare. Obiectul JSON obținut este criptat folosind algoritmul definit în header și alipit șirului de caractere ce conține reprezentarea headerului. La final se atașează semnătura, ce autentifică destinatarului identitatea emițătorului, desigur criptat în același algoritm. Astfel se obține un șir de caractere ce încorporează obiectul JSON cu afirmațiile interlocutorilor.

* + 1. Hibernate

Un ORM (Object Relational Mapper) este un framework ce permite maparea tabelelor din baza de date relațională la entități existente în aplicație, astfel încât dezvoltatorii să poată accesa baza de date cu un minim de efort, fără a mai fi necesar configurarea conexiunilor sau driverelor, deschiderea/închiderea de conexiuni și scrierea de cod SQL sub formă de șir de caractere. Hibernate este o soluție ORM pentru limbajul Java, având la bază JPA (Java Persistence API), preluând modalitatea de încărcare, stocare și accesare a datelor. În cadrul aplicațiilor web, accesarea datelor se face folosind un obiect special numit DAO ( Data Acces Object), totalitatea obiectelor formând Data Access Layer. Hibernate preia metodele descrise în DAO transpunându-le în cereri SQL, tot el coordonând conexiunea la baza de date, maparea și instanțierea obiectelor, oferind dezvoltatorului obiectul/obiectele deja create. Maparea entităților se face cu ajutorul adnotării @Entity ce are ca parametru numele tabelului ce este reprezentat de entitate, iar fiecare câmp având adnotarea @Column și numele coloanei ca parametru. Se pot omite parametrii din ambele adnotări dacă numele obiectelor și ale atributelor sunt aceleași cu ale reprezărilor din baza de date.

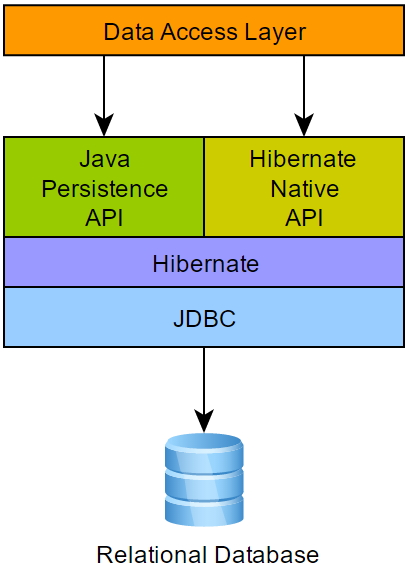


Fig 5. Arhitectura nivelului datelor a unei aplicații ce folosește Hibernate[[34]](#footnote-24)

În Hibernate cheile primare sunt marcate prin adnotarea @Id. Comportamentul lor este determinat de adnotarea @GeneratedValue, parametrul ”generation” determinând felul în care se generează cheia primară, având valoarea ”SEQUENCE”, automat necesită un parametru suplimentar în care se specifică numele secvenței, la generarea unui nou obiect, Hibernate apelează secvența și atribuie obiectului cheia primară generată. Parametrul ”IDENTITY” indică unicitatea cheii primare și generarea ei automată de către baza de date. La adăugarea unui nou obiect, după salvare, atributul obiectului ce simbolizează cheia primară este actualizat. Cu valoarea ”AUTO” Hibernate va decide ce strategie va aborda în funcție de baza de date folosită.

Tablele legate în baza de date au o reprezentare și în interiorul entităților Hibernate. La aducerea unui obiect din bază, se pot creea pe lângă tipurile primare și alte obiecte ce reprezintă tabele. Pentru a se face legătura corect, trebuie specificat relația dintre cele două tabele folosind adnotările @ManyToOne, @OneToOne, @OneToMany și @ManyToMany alături de specificarea coloanei ce reprezintă cheia străină prin adnotarea @Column și parametru numele coloanei.

Java Persistence API oferă o interfață pentru acesarea obiectelor, aceasta poartă numele de Crud Repository (Create Read Update Delete), ce înglobează o serie de operații asupra bazei de date pe baza cheii primare. Spring a preluat această interfață a extins-o și a adăugat o serie de funcționalități obținând astfel Spring Data.

Spring Data facilitează crearea obiectelor DAO, reducând timpul de dezvoltare, implementarea efectivă de cereri simple nefiind necesară. La baza compunerii cererilor simple se află convenția de nume a metodelor declarate în interfață, de exemplu o metoda ”User findByUsername(String username);” va genera o cerere SQL de tipul ”WHERE username = &username”. Aceste cereri pot deveni mai complexe pentru obiectele compuse, permițând căutarea după atribute ale obiectelor ce compun obiectul de repository.

1. Proiecte similare

Internetul a ajutat dezvoltarea companiilor prin simplificarea administrării angajaților, fie prin aplicații de tip CRM, rețele de tip Intranet, aplicații web de task management sau aplicații interne. Unul dintre principalii beneficiari al acestor tipuri de aplicații este Departamentul de Resurse Umane, prin intermediul aplicațiilor web s-a redus cantitatea de documente fizice necesare iar căutarea și indexarea acestora s-a făcut mult mai ușor și mai centralizat.

În cadrul companiilor cu mulți angajați, obligația de completare și întreținere a datelor cu caracter personal s-a trecut din grija departamentului HR în grija angajatului însuși, având ca instrument diferite aplicații de cele mai multe ori web ce îi permit accesa baza de date și îi permit actualiza și completa diferite date. Astfel se crește eficiența departamentului de HR, se reduce timpul necesar actualizării datelor și cantitatea de documente fizice necesare. Acest tip de aplicații numit și Employee Self Service permit structurarea și afișarea informațiilor precum profilul de lucru, adresa de reședință, asigurarea medicală și oferă servicii precum actualizarea acestor informații și administrarea zilelor de concediu.

Companiile ce nu au implementată o astfel de platformă folosesc un template Excel partajat între angajații unui departament sau între membrii unei echipe, aceasta fiind una dintre cele mai întâlnite metode de administrare a concediilor. Deși o metodă ieftină și rapidă, template-ul Excel este depășit în cazul unei echipe mai numeroase, iar în cadrul unei întregi companii nu este modelul recomandat. De asemenea îi lipsesc numeroase facilități dorite de client precum persistarea datelor întro bază de date, integrarea directă cu alte sisteme, securitatea datelor, toate acestea compunând o bază nocivă folosirii acestui model.

Un alt model întalnit este acela al aplicațiilor desktop. Deși rezolvă problemele template-ului Excel, aplicația desktop are câteva deficite. În primul rând e dependentă de sistemul de operare, facilitatea aplicației de a migra de la un sistem de operare la altul variază în funcție de limbajul în care este scrisă. O altă dependență este cea de installer, pentru a putea folosi aplicația un utilizator trebuie să o instaleze pe dispozitivul folosit, aducând cu sine alte probleme precum insuficiența drepturilor utilizatorilor, necesitatea unei echipe de IT suport ce poate să instaleze aceste aplicații, eventualele documente suplimentare necesare oferiri de drepturi utilizatorului și altele. Dependența de dispozitivul folosit se manifestă dacă apare cerința dezvoltării unei aplicații mobile, întreaga aplicație necesitând a fi reproiectată. Actualizările aplicațiilor desktop pot aduce diferite probleme, neconcordanțe cu sistemele de operare folosite, bug-uri de mediu și altele.

Altă modalitate de administrare a concediilor este prin intermediul aplicațiilor web. Acestea nu depind de un installer, pentru a le folosi fiind necesară doar conexiunea la internet. Securitatea aplicației ce nu exista în cadrul template-urilor Excel acum se realizează pe baza unor identificatori, deobicei un nume de utilizator și o parolă. Aplicațiile web sunt portabile spre deosebire de aplicațiile desktop, orice dispozitiv, indiferent de sistemul de operare folosit, atâta timp cât suportă un browser cu facilități minime determinate putând fi folosit spre utilizare. Pentru trecerea la dispozitivele mobile,efortul de dezvoltare este minim, interfața fiind singurul modul în care sunt necesare posibile modificări dar cu apariția noilor framework-uri de CSS, interfața păstrează în mod automat o structură logică precisă indiferent de dimensiunea ecranului. Are aceleași facilități ca și o aplicație desktop dar pe deasupra oferă mecanisme suplimentare de securitate și întreținere, facându-le un model eficient pentru dezvoltarea aplicațiilor de management al concediilor. Pe piață există aplicații ce oferă prin diferite metode posiblitatea administrării concediilor angajaților într-o companie.

* 1. Employee Vacation Tracking Calendar[[35]](#footnote-25)

Template-ul oferă posibilitatea administrării cererilor de concediu a câtorva angajați. Administrarea acestora se face de către manager sau secretară, fără a putea oferi facilități suplimentare, propunându-și înlocuirea gestionării prin intermediul calendarelor fizice, permițând actualizarea automată la inserare. Nu se pot oferi detalii suplimentare pentru fiecare cerere în parte, cererile sunt persistate doar în interiorul documentului. Dacă documentul se face public și fiecare angajat își completează cererea, template-ul nu poate oferi securitatea și siguranța unei aplicații. O altă posibilitate este efectiv printarea template-ului și completarea manuală a angajaților, dacă administratorul preferă această abordare. Template-ul este gratuit, oferă facilități minime de gestionare, în acesta regăsindu-se toate beneficiile dar și insuficiențele abordării prin template Excel. Acest model este indicat firmelor de dimensiuni reduse,ale căror angajați nu au neapărat cunoștințe de programare singurul său scop fiind acela de a structura și organiza cererile acestora.

* 1. Timetastic[[36]](#footnote-26)

Aplicație web ce permite administrarea,centralizarea și stocarea cererilor de concediu, oferă utilizatorilor o multitutdine de facilități. Pentru a efectua o cerere de concediu se poate folosi calculatorul sau telefonul mobil, cererea fiind distribuită managerilor. Aceștia printr-un click pot accepta sau respinge cererile, iar utilizatorii pot vedea automat starea cererilor lor. Comunicarea cererii dintre angajați și administratori se face prin intermediul emailului, la crearea unei cereri de concediu administratorii sunt notificați iar în urma deciziei utilizatorul primește un email cu starea cererii sale. Alte facilități descrise în site-ul lor de prezentare sunt posibilitatea definirii de departamente și desemnarea administratorului de concedii pe acel departament, crearea de tipuri diferite de concediu, în funcție de tipul companiei și nevoile acestora. De asemenea Timetastic este integrat cu Outlook și Gmail, oferind posibilitatea administrării și afișării perioadelor de concediu în calendarele specifice fiecărei aplicații. Se mai pot crea și administra seturi de zile libere în funcție de diferite politici și multe alte facilități. Serviciile aplicației sunt contra cost,prețul abonamentului fiind determinat per utilizator per lună, informațiile foștilor angajați persistă în interiorul aplicației sub forma unei arhive, doar utilizatorii activi fiind taxați.

* 1. WhosOFF[[37]](#footnote-27)

Câștigătoare a premiilor ”Software Satisfaction Award” în anul 2012 în Marea-Britanie, aplicația web WhosOff este ce-a mai populară soluție pentru administrarea concediilor. Având o vechime de 10 ani în domeniu, cu peste 100.000 de utilizatori anual și cu peste 3 milioane de cereri de concediu procesate[[38]](#footnote-28), această experiență oferită prin intermediul feedback-ului utilizatorilor a dat posibilitatea dezvoltării unei aplicații mature și flexibile ce se adaptează cerințelor firmei. Oferă posibilitatea administrării cererilor de concediu într-un mediu centralizat, disponibil permanent, indiferent de numărul de angajați și poziționare geografică. Datorită structurii sale de aplicație web nu necesită instalare de software, singura necesitate este un dispozitiv conectat la internet și un browser web. Aplicația este sigură, oferind securitate, fiind certificată ISO27001 iar datele au copii de rezervă în servere speciale înafara rețelei folosite de aplicație. Aplicația mobilă este disponibilă pe 3 sisteme de operare, BlackBerry, Android și iOS, permițând angajaților să ceară concediu și din afara biroului. Oferă posibilitatea extragerii de rapoarte, pentru unul sau mai mulți utilizatori și expunerea lor sub format CSV. Zilele de concediu sunt administrate granular prin intermediul sistemului de restricții, eficientizând și evitând crearea de perioade goale în care prea mulți angajați lipsesc. În plus față de Timetastic, permițând administrarea cererilor de concediu în masă. Echipa de dezvoltare oferă posibilitatea modificării aplicației, în schimbul unei taxe mai mari, branding-ul companiei poate fi integrat în aplicație alături de alte sisteme. Aplicația este contra cost, prețul fiind determinat de numarul de angajați din companie, începând de la 12.50 lire pentru o companie de 10 angajați cu prețul de 1.25 lire per angajat la 3763.50 lire pentru o companie de 5000 de angajați cu prețul de 0.75 lire per angajat.

Pe lângă Timetastic și WhosOff ce oferă servicii contra cost există numeroase aplicații ESS specializate, interne, ce permit printre altele managementul concediilor, satisfăcând diversele cerințe impuse de profilul aplicației și de dorința utilizatorilor. Câteva exemple de unități ce posedă astfel de aplicații : Universitatea din Washington[[39]](#footnote-29), Universitatea din Minnesota[[40]](#footnote-30), John Deere[[41]](#footnote-31) și altele.

Modelul folosit pentru administrarea concediilor depinde de numeroși factori, pritre aceștia fiind dimensiunea firmei, profilul acesteia, gradul de instruire a angajaților, gradul de globalizare al firmei, diferiți factori ce țin de politica firmei și de regulile de administrare ale țării gazdă. Altă decizie importantă este modul de achiziționare al aplicației, dacă se dorește o aplicație generică sau dezvoltată intern. Această decizie depinde de particularitatea cerințelor utilizatorului, de fondurile alocate rezolvării acestei probleme, firmele mici cu buget redus tinzând să renunțe la particularitățile dorite din partea aplicațiilor în detrimentul unor platforme generice dar mai ieftine. O altă variabilă este reprezentată de timpul dorit până la punerea în producție, dezvoltarea unei aplicații interne necesitând o perioadă mai lungă de timp decât achizitionarea și configurarea unei aplicații deja dezvoltate. Aplicațiile interne sunt instabile, necesitând după punerea acestora în producție o perioadă de timp de maturizare, când se descoperă și se rezolvă majoritatea bug-urilor. De luat în calcul este și perspectiva de viitor, aplicațiile generice rămânând relativ statice pe parcursul perioadei de viață, eventualele adăugiri de caracteristici noi facându-se doar dacă suficienți utilizatori o cer în comparație cu aplicațiile specifice unde orice caracteristică nouă poate fi pusă în producție cât mai curând.

1. Implementare

În urma analizei clientului, firma de dezvoltare software WhiteBox Technologies s-a observat un efect negativ în cadrul administrării cererilor de concediu datorită creșterii efectivului de angajați. S-a ajuns la concluzia că modelul actual al administrării cererilor de concediu nu mai este eficient fiind dificil de administrat. Modelul actual presupunea transmiterea manuală a cererilor de concediu sub format Microsoft Word prin intermediul emailului și notarea manuală de către manager a perioadelor de concediu efectuate fără o centralizare a acestora, calculul zilelor de concediu făcându-se folosind cererile anterioare.

Pornind de la analiza efectuată clientul a adus o listă de cereri :

1. Automatizarea procesului depunerilor cererilor de concediu
2. Securitatea procesului de administrare și vizualizare conform celor două mari principii din securitate, autentificarea și controlul accesului.
3. Notificarea automată a managerilor în urma creeri unei cereri de concediu și notificarea utilizatorului ce a depus cerererea cu statutul ei în urma aprobării sau respingerii.
4. Automatizarea procesului de generare a documentelor Microsoft Word și atașarea acestora în notificare
5. Vizualizare facilă și calculul automat al zilelor de concediu rămase ținând cont de zilele libere declarate oficial
6. Posibilitatea administratorului de a vizualiza detaliile utilizatorilor și a calendarului cererilor de concediu
7. Posibilitatea administrării datelor cu caracter personal de către utilizatori
8. Integrarea cu aplicația Slack pentru gestiunea și distribuirea de ”knowledge sharing”

Pe rând au fost elimintate soluțiile ce nu se potriveau cerințelor clientului, ajungându-se într-un final să se aleagă dezvoltarea internă a unei aplicații web.

Soluția prin folosirea de template Microsoft Excel a fost eliminată datorită cerinței de centralizare, automatizare și gestionare a cererilor de concediu acest model nesatisfăcând nici-una dintre cerințe.

Datorită avansării tehnologiilor actuale în domeniu aplicațiilor web, a apropierii ca structură și design tehnic de cele desktop dar păstrând flexibilitatea și ușurința de accesare a paginilor web s-a decis dezvoltarea aplicației web în detrimentul celei desktop.

Alegerea unei aplicații generice s-a dovedit ineficientă datorită costului ridicat al acestora per angajat deși având un număr de angajați suficient de mare pentru a nu mai putea fi folosit vechiul model, este totuși mic comparat cu numărul de angajați ale unor firme de tip multinaționale. Profilul firmei îi permite dezvoltarea de aplicații interne nefiind necesară subcontractarea unei echipe să dezvolte aplicația, angajații deținând aptitudinile necesare dezvoltării de aplicații web. Alte proiecte nu pot rezolva în totalitate problema datorită specificității cerinței de dezvoltare, a integrării cu alte sisteme și a regulilor de acordare a concediilor specifice spațiului românesc.

Proiectul iși propune ca în versiunea finală să permită administrarea și generarea de cereri de concediu automat, să permită utilizatorilor să iși calculeze zilele de concediu mai ușor și managerii să poată administra cererile angajaților, calculul zilelor de concediu să fie specific țării în care activează clientul și să se poată integra cu aplicațiile deja folosite. O parte din mecanismele și facilitățile oferite de noua aplicație vor fi preluate de la aplicațiile din domeniu deja existente precum mecanismele de tratare a cererilor și automatizarea lor, trecându-se prin stări similare, alături de mecanisme noi, specifice ce vor satisface cerințele clientului.

* 1. Modelul de analiză al aplicației

În urma colaborării cu clientul s-a obținut un model de analiză al datelor, un model rudimentar ce suprinde principalele funcționalități ale aplicației. Deși nu ilustrează implementarea efectivă a aplicației, acesta oferă informații cu privire la legătura dintre entitățile existente în aplicație, clarificând diferite aspecte legate de implementare.

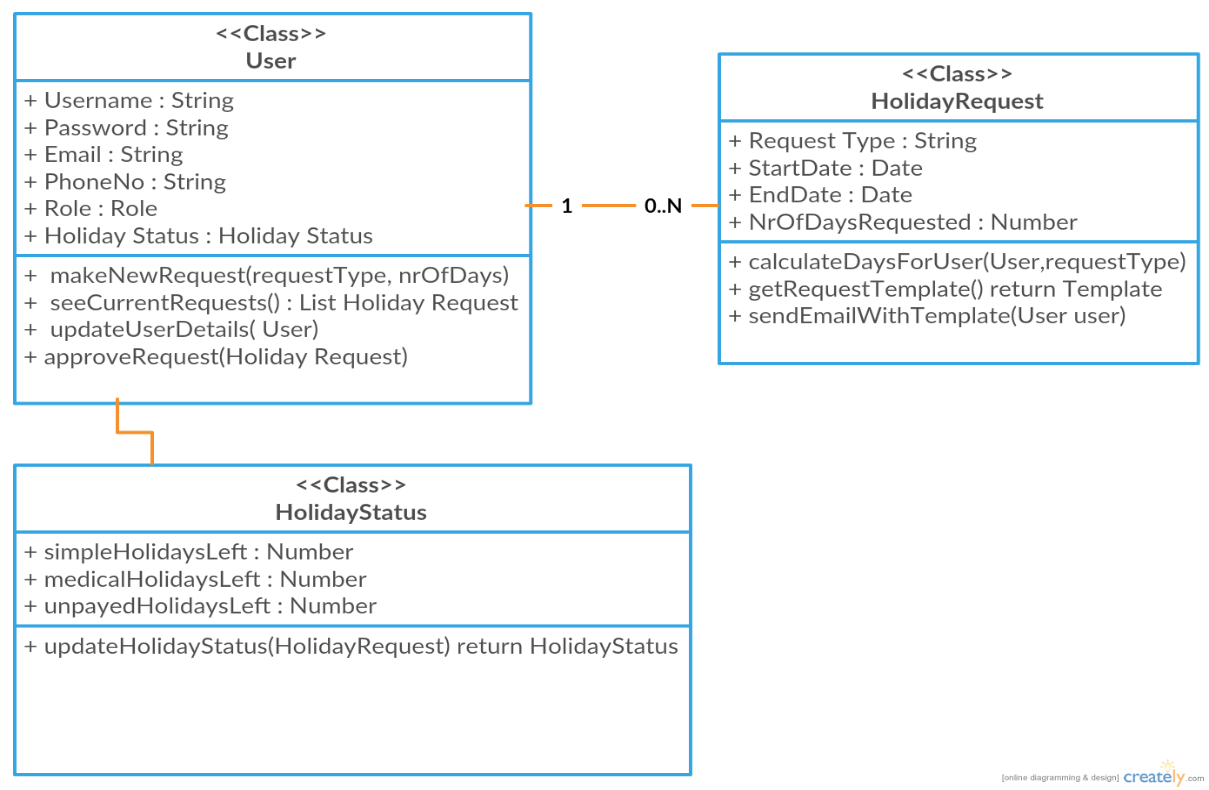


Fig 6.Modelul de analiză al aplicației (Prototip)

Astfel se definește entitatea User, utilizatorul aplicației, actorul ce se autentifică în pagina de start. În această clasă se țin credențialele de logare, rolul utilizatorului în cadrul aplicației, un obiect în care se va ține situația actuală la nivel de zile de concediu și alte detalii personale precum email și număr de telefon. În funcție de rolul utilizatorului acesta poate să creeze noi cereri, să vadă cererile deja existente și să-și actualizeze informațiile personale în cazul utilizatorilor normali și în plus să aprobe cereri în cazul administratorilor.

Relația dintre utilizator și detaliile legate de situația zilelor de concediu este de unu la unu, în clasa HolidayStatus persistându-se pentru fiecare tip de concediu numărul de zile existente. Folosind Inversion of Control, prin clasa HolidayStatus ar trebui să se actualizeze zilele respective în urma acceptării unei cereri de concediu.

Fiecare utilizator are atașată o listă cu cererile aprobate, afișate utilizatorului în interfață. Tot aceeași clasă se folosește și în cazul cererilor neaprobate, rolul acestora fiind de a permite generarea documentului ce reprezintă cererea cât și trimiterea de emailuluri utilizatorilor ce au legătură cu noua cerer de concediu.

Deși pe parcurs modelul datelor a devenit mult mai complex, fiind adăugate pe parcurs alți parametrii, anumite metode fiind mutate de la o entitate la alta, chiar adăugate entități suplimentare, în esență legăturile între cele trei mari entități, utilizator-cereri-stare cereri au rămas aceleași. În paragrafele următoare voi lua câte o cerință a utilizatorului și voi descrie metoda de implementare.

* 1. Detalii de implementare
     1. Securitatea aplicației

Conform cerinței utilizatorului aplicația trebuie să fie sigură, permițând controlul atât la nivel de autentificare cât și asupra controlului accesului oferind utilizatorilor siguranța asupra datelor chiar și în cazul unei breșe de securitate. Pe de altă parte se dorește un nivel minim de intruziune în experiența de folosire a aplicației pentru utilizatori.

Controlul la autentificare se face prin intermediul credențialelor de autentificare, prin nume de utilizator și parolă. În baza de date se persistă pentru fiecare utilizator în parte numele de utilizator și o parola criptată prin algoritmul SHA-256. Acest algoritm de mapare are o caracteristică importantă, aceea că folosind aceleași date de intrare vom obține de fiecare dată același șir de caractere și orice modificare oricât de mică în datele de intrare va produce un șir complet diferit. Deși la prima vedere această abordare pare destul de sigură, există un tip de atac la care este vulnerabilă, atacul prin tip Reverse Lookup Table. Atacul presupune, cunoscând algoritmul de criptare, folosirea unei baze de date în care se păstrează cuvinte uzuale criptate în algoritmul cunoscut. Dacă baza de date este spartă, atacatorul poate afla parolele utilizatorilor și admițând că utilizatorii obișnuiți folosesc aceeași parolă în mai multe contexte, îi face vulnerabili la și mai multe atacuri. Eliminarea acestui pericol se face prin intermediul salt-ului, un șir de 32 de caractere pseudoaleatoare, considerat sigur din punct de vedere al securității. În baza de date se persistă în final parola utilizatorului concatenată cu salt-ul și criptată prin SHA-256 impreună cu numele utilizatorului și salt-ul singur. Procesul de autentificare presupune preluarea parolei introduse în interfață, concatnarea cu salt-ul din baza de date și criptarea lui. Dacă șirul criptat este identic cu șirul păstrat în bază, utilizatorul este considerat autentficat, în caz contrar se generează eroare. De observat folosirea tiparului de design de tip Singleton la nivel de aplicație datorită timpilor de inițializare relativ mari a claselor MessageDigest și SecureRandom oferite de pachetul Java Security, folosite în cazul criptării și autentificării utilizatorilor.

Pentru a limita gradul de intruziune la minim, cerând o singură dată utilizatorilor să se autentifice, după ce sunt acceptați și autentificați se generează un token de autentificare de tip JSON Web Token ce atestă validitatea utilizatorului. În token se păstrează o serie de afirmații (claims) precum numele utilizatorului, și data la care a fost creat, dată ce este folosită la determinarea expirării tokenului, momentan având valoarea de 60 de minute. În partea de interfață, tokenul este păstrat folosind facilitatea oferită de HTML5, localStorage, la fiecare cerere către server adăugându-se acest token ori în header în cazul cererilor de tip POST sau ca parametru de URL în cazul cererilor de tip GET. În coloanele tabelului Users se află și momentul ultimei actualizări,pentru a determina dacă tokenul folosit de utilizator chiar dacă nu este expirat reflectă cu strictețe starea reală a datelor utilizatorului, la crearea obiectului de autentificare se compară data creeri tokenului cu data ultimei actualizări, dacă tokenul este mai vechi se va genera un token nou.

Folosind facilitatea oferită de Spring Security de a înlănțui filtrele, am creat un filtru special folosit la extragerea de informații din token, crearea unei autentificări (Authentication) în cazul în care aceasta nu există în contextul de securitate Spring și trimitrea către urmatorul filtru.

Securitatea resurselor la nivel de URL am realizat-o prin intermediul Spring Security Configuration, în funcție de modelul de URL diferite reguli. Oferă un control mai bun asupra resurselor, permițând accesul la acestea doar utilizatorilor autentificați, adăugarea de noi filtre și configurarea detaliilor de securitate legate de sesiune.

În cadrul aplicației am dezactivat opțiunea de Cross Site Request Forgery, optiune cu care Spring vine deja porintă. Ea nu este necesară în primul datorită stării la nivel de sesiune, fiecare sesiune este stateless, alt motiv este lipsa cookie-urilor, autentificarea se face pe baza tokenului în interiorul header-ului nu prin cookie. Alt motiv este lipsa cererilor directe asupra serverului, toate cererile se fac prin intermediul JavaScript și AJAX, tipul datelor fiind verificat înainte. Deși utilă în cazul aplicațiilor multipage, CSRF în acest context nu aduce decât overhead de calcul pentru token-ul de sincronizare și trafic suplimentar în transferul lui.

Altă măsură de securitate este trecerea sesiunilor în politica stateless. Îm aplicațiile single page de tip MVC se pune accent mai mult pe partea de client, de anumite responsabilități ocupându-se acesta, cum ar fi păstrarea tokenului de autentificare, la fiecare cerere clientul oferind tokenul iar serverul răspunzându-i după ce îl validează. Cookie-urile sunt înlocuite de localStorage, evitându-se în acest fel inconvenientele aduse de acestea.

Resursele păstrate pe server precum fișierele html, fișierele JavaScript sau cele CSS nu necesită autentificare, încărcarea lor facându-se prin cereri de tip GET, ar trebui atașat tokenul de autentificare în ele, acest lucru fiind dificil de implementat. Calea declarata ”/auth” este calea către pagina de login, pentru aceasta deasemenea nefiind necesară autentificarea. Pentru orice alt URL mapat în schimb este necesară autentificarea ori folosind un token, sau în caz contrar obținerea unuia în urma autentificării manuale.

Pentru a putea realiza controlul accesului, fiecărui utilizator este asociat câte un rol, fie rolul de administrator (Admin) fie rolul de utilizator obișnuit (User). Aceste roluri păstrate în baza de date într-un tabel special numit Roles, permit restricționarea accesului la resurse nu numai pe baza autentificării ci și pe baza rolului acestuia. În obiectul Authentication Spring oferă posibilitatea asocierii a mai multe roluri unui utilizator pentru a permite accesul la resurse mai granular, dar în contextul aplicației curente nu este necesar decât un singur rol. Transformarea dintre un obiect User obținut prin interogarea bazei de date și un user autentificat menținut în contextul Spring se face folosind template-ul Factory.

* + 1. Automatizarea procesului de administrare a cererilor

Odată autentificat, un utilizator poate să folosească facilitățile aplicației fără a mai fi necesari pași suplimentari. Pentru a putea depune o cerere nouă de concediu, el trebuie să navigheze cu ajutorul ”navbar”-ului din partea de sus a ecranului. În urma click-ului se generează un eveniment ce este tradus prin apelul funcție JavaScript navbarGetRequest(activeElement, url). Pentru a crește eficiența, a reduce comunicarea excesivă cu serverul și datorită lipsei necesității actualizării paginilor principale la fiecare cerere, acestea sunt stocate în pagină în interiorul unor elemente de tip ”div”, ascunse utilizatorului. În funcția amintită anterior se verifică existența acestor elemente ascunse, dacă ele există se vor afișa fără a mai face un apel către server, dacă nu există se va apela serverul pentru a prelua și actualiza conținutul dorit marcat prin primul parametru al funcției, ”activeElement”.

Așadar în urma navigării către fereastra de cerere nouă de concediu se verifică existența conținutului și în absența lui se generează o cerere AJAX de tip GET, ce va primi text de tip HTML, browserul interpretându-o ca atare, i se atașează în header tokenul de autentificare și este postată. Pe partea de server cererea este prinsă de obiectul Dispatcher Servlet, servletul generic oferit de Spring și este trecută prin filtrele Spring Security. Se verifică existența obiectului de autentificare în context, în cazul în care lipsește se creează folosind datele transmise prin token. După procesul de verificare a autenticității, Spring mapează URL-ul cu o metodă din cadrul controllerelor declarate. Fiind o cerere simplă se compune formularul necesar depunerii unei noi cereri cu ajutorul Thymeleaf și se întoarce fragmentul dorit. În urma răspunsului cu succes din partea serverului, pe partea de client se actualizează conținutul și se adaugă în document noul element ascuns ce ține formularul pentru o eventuală refolosire.

Datele necesare compunerii unei noi cereri sunt data de început, data de sfârșit, tipul concediului și o eventuală descriere. Pentru selectarea perioadelor de început și sfârșit am folosit un framework gratuit numit Monthly Js[[42]](#footnote-32) ce încorporează atat facilitatea de ”date-picker” cât și cea de calendar de evenimente. Folosind date-picker se evită erorile de formatare a șirului de date ce reprezintă perioada și oferă utilizatorului o formă interactivă de a selecta această perioadă. Tipul concediului determină comportamentul ulterior al cererii, primele două tipuri fiind relativ similare, concediul de odihnă și concediul fără plată dar ultimul, concediul medical are o serie de facilități distincte, primele două tipuri fiind analizate împreună iar concediul medical va fi analizat separat.

Odată completată cererea urmată de submiterea formularului, evenimentul este prins în interiorul fișierului submitHandlers.js pentru a fi compusă cererea. Pentru concediul de odihna/ fără plată din interfață se preiau inputurile , se crează un obiect și se transpune în format JSON. Asemeni cererilor de tip GET, se adaugă în header tokenul de autentificare, în secțiunea de date datele formatate și se postează cererea. După validarea autenticității tokenului și trecerea prin filtrele Spring, în controller se compune un obiect de tip HolidayPeriod folosind patternul Builder ce transformă cererea din interfață în obiect model, adăugând și informații suplimentare precum numărul de zile de concediu efective. Calculul acestor zile se face ținând cont de zilele libere ce variază în funcție de an, zilele libere statice și sfârșitul de săptămână. Păstrate în baza de date în tabela ”Holidays” zilele libere se împart în două categorii, statice și dinamice, diferența dintre cele două fiind marcată de câmpul ”year”, pentru zilele statice având valoarea 0. La calculul numărului de zile efectiv din cerere se preiau din bază toate zilele declarate libere din anul curent împreună cu toate zilele statice. Am ales această abordare datorită toleranței la modificare, zile libere fiind declarate și scoase relativ des, dar în total numărul lor nu este mai mare de 20. Perioada de start date și end date este separată într-o listă de zile, fiecare zi este parcursă, se verifică dacă este sâmbătă sau duminică, în cazul contrar se verifică dacă face parte din lista de zile libere declarate oficial. Dacă ziua nu este declarată liberă și nici nu face parte din zilele de weekend se incrementează un contor ce la final va ține numărul de zile de muncă efectivă ce vor fi cuprinse în perioada de concediu. După compunerea obiectului, noua cerere este introdusă în baza de date și se trimite un email către toți administratorii din aplicație în vederea notificării acestora.

Utilizatorii ce au rolul de administrator au interfața diferită față de utilizatorii obișnuiți rolul administratorilor fiind acela de aproba sau respinge cererile, de a vedea starea tuturor angajaților și a interoga calendarul de evenimente. În urma notificării noii cereri prin email cu atașament, în atașament punându-se cererea fizică completată, administratorul trebuie să navigheze în pagina de cereri unde îi sunt listate toate cererile neaprobate depuse de toți utilizatorii. Selectarea cererilor spre aprobare sau respingere se face print intermediului bifelor, acestea putând fi selectate și în masă prin bifa din capul de tabel. Odată submis formularul cu lista de identificatori unici pentru cererile selectate spre aprobare în format JSON și validarea autentificării se actualizează statutul fiecărui user.

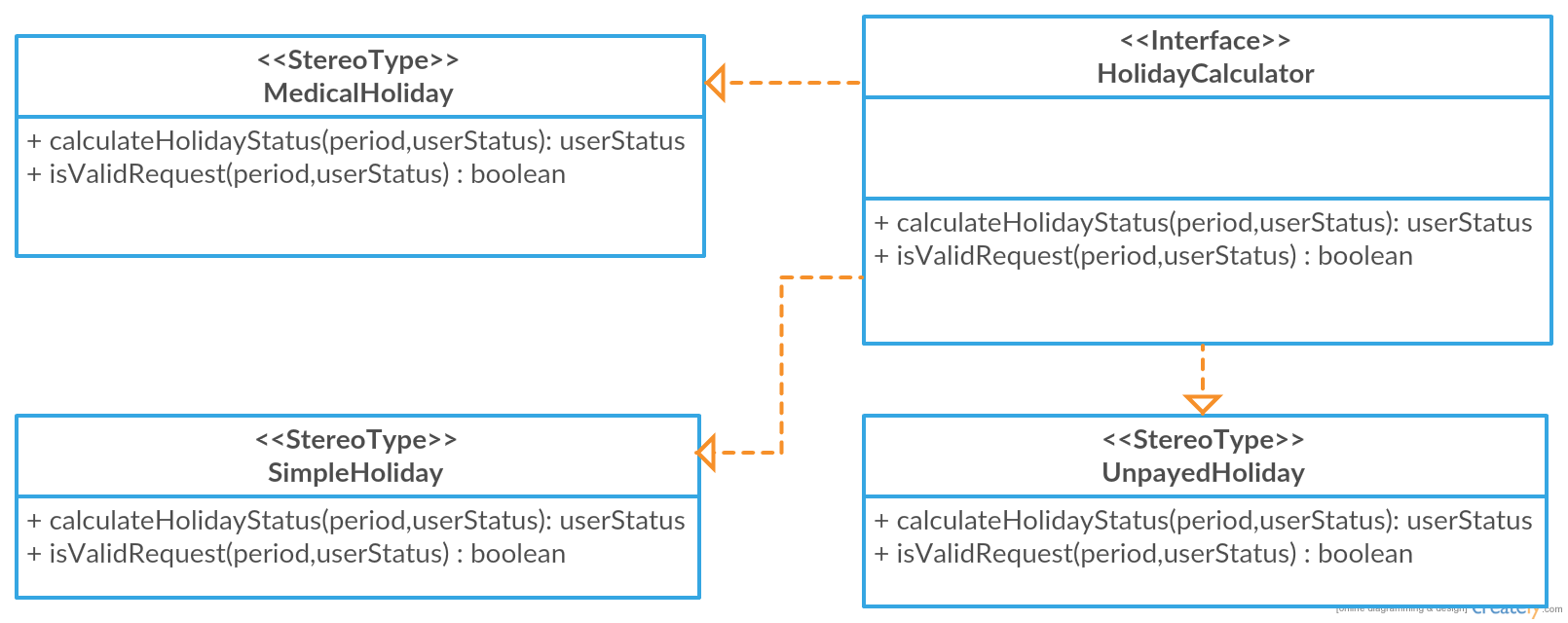


Fig. 7 Structura obiectelor ce actualizează zilele de concediu ale utilizatorilor

În câmpul ce ține tipul cereri este folosit ulterior în calculul zilelor rămase de concediu prin intermediul Java Reflection API, numele tipului cererii fiind identic cu numele claselor Java ce implementează interfața HolidayCalculator. Această abordare am folosit-o deoarece am dorit evitarea structurilor monolitice permițând adăugarea ulteriorară de noi tipuri de concediu, fără a impacta structura aplicației. Cum se poate observa în fig 7 aceste clase implementează două metode, prima ce actualizează zilele de concediu conform regulilor de acordare, iar a doua ce determină dacă cererea este validă sau nu. Regulile de actualizare a zilelor de concediu diferă în funcție de tipul cererii, pentru zilele de concediu de odihnă raportându-se zilele din anul precedent, la calculul acestora întâi scăzându-se din zilele din anul trecut, urmând cu zilele din anul curent, la celelalte tipuri de concediu se scad din cele 90 de zile permise de legea în vigoare. Deși Java Reflection API este o procedură destul de nesigură și periculoasă, eventualele pericole precum instațierea altor obiecte decât cele asupra cărora un utilizator are dreptul a fost evitată prin exilarea acestei structuri într-un pachet special, extragerea claselor facându-se specificând calea către pachet. Dacă se modifică fraudulos tipul cererii , administratorul va putea observa în interfață cererea modificată și o va respinge. După actualizare se generează un email ce va fi trimis către persoana ce a depus cererea notificând-o de aprobarea sau respingerea ei, urmat de salvarea întregii liste cu noile statuturi ale zilelor de concediu ale utilizatorilor în baza de date.

Concediul medical este diferit de celelalte două tipuri datorită cererii de atașare a unui fișier de tip media, imagine sau pdf ce reprezintă ori certificatul de concediu medical primit de la medicul de familie, ori fișa de externare în cazul intervențiilor în regim de urgență. Altă deosebire este posibilitatea alegerii de perioade în trecut, dacă pentru concediul de odihnă și concediul fără plată nu se pot selecta perioade din trecut, în cazul concediilor medicale se poate datorită internărilor în regim de urgență. La selectarea din drop down a concediului fără plată, în formularul din interfață apare un câmp nou, un input de tip fișier în care utilizatorul să adauge documentele enumerate anterior. Tipul cererii AJAX se schimbă , în datele trimise către server nu se mai trimite formularul în format JSON ci se trimite obiectul definit in HTML5, FormData la care se atașează în cheia ”medicalFile” inputul de tip fișier , iar în cheia ”period” se atașează restul intrărilor din interfață. Pe partea de server se construiește cererea folosind același Builder Pattern, dar printr-o nouă metoda ”getMediaHolidayPeriod” ce primește datele cererii în format JSON și fișierul de tip MultiPart. Fișierul este păstrat în baza de date într-un tabel special numit ”attached\_files” în care se reține numele fișierului, tipul MIME și biții ce formează fișierul. Datorită dimensiunii reduse a fișierelor, ele pot fi păstrate în baza de date chiar în interiorul tabelului într-o coloană de tip medium BLOB. Fișierul atașat este legat de cerere prin intermediul cheii străine attached\_file\_id și salvată în baza de date. Descărcarea fișierelor atașate alături de descărcarea documentului Word se poate face la vizionarea detaliilor cererii. Ambele cereri de descărcare presupun deschiderea într-o fereastră noua prin funcția JavaScript ”window.open” ce face un apel de tip GET către server folosind url-ul trimis ca parametru. În acest tip de apel nu se pot pune headere din acest motiv, pentru a păstra securitatea accesului la documente, tokenul de autentificare se trimite ca parametru de URL. Este totuși o metodă nu foarte sigură datorită expunerii în mod direct a tokenului prin intermediul istoricului cererilor utilizatorului, acestea rămânând în browser o perioadă îndelungată. La nivelul filtrului Spring se încearcă extragerea tokenului din headers dar cum nu se poate ,se încearcă extragerea acestuia din parametrul de URL apoi este validat. Pentru a determina identificatorul cererii ce are atașate fișierele se folosește o variabilă de cale ce este preluată din interfață. Cererea adusă din baza de date împreună cu fișierul atașat prin intermediul HttpServletResponse se preia outputStream și se copiază biții fișierului persistat. Se setează tipul MIME al răspunsului , ”application/octet-stream” ce indică browserului că răspunsul este un fișier și nu o pagina HTML , în header se indică numele noului fișier și se trimite răspunsul către client unde se începe descărcarea.

* + 1. Transmiterea de email

O altă cerință specificată de client este transmiterea de email-uri între aplicație și utilizatorii săi pentru a îi notifica de eventualele cereri sau alte modificări. Pentru transmiterea emailurilor se foloseste biblioteca oferită de java, Javax Mail ce permite crearea de emailuri cu conținut text și atașarea de fișiere la acestea.

Emailurile se construiesc dinamic, în baza de date păstrându-se conținutul text al emailului în tabelul ”emails” având ca cheie primară tipul emailului alături de header și conținutul text. Pentru particularizare am notat elementele variabile cu un pattern de text : ”?NUMEVARIABILA”. Folosind acest pattern, extrag din baza de date template-ul emailului și completez variabilele prin manipularea șirului de caractere. Obiectul MimeMessage permite atât adăugarea de conținut text cât și atașarea de documente, fiind un obiect mixt, pentru fiecare fișier în parte se crează câte un obiect de tip MimeBodypart căruia i se atașează un DataHandler care se ocupă de legarea fișierelor de mail. Situațiile în care se cere trimiterea de email sunt la aprobarea sau respingerea unei cereri de către un administrator, la depunerea unei cereri către toți administratorii și la crearea unui nou cont prin care se notifică noul utilizator și i se înmânează parola temporară.

* + 1. Completarea documentului Word

O altă cerință impusă de client este automatizarea și păstrarea într-o oarecare măsura a vechii metode de comunicare a cererilor de concediu prin intermediul documentelor de tip Word. Pentru a mă conforma am decis folosirea unei biblioteci open source de Java ce permite manipularea obiectului Microsoft Open XML ce stă la baza fișierelor de tip .docx, .pptx si .xslx. Are la bază JAXB ce construieste obiecte Java și le încarcă în memorie oferind posibilitatea modificării programatice a obiectului, modificări ce se reflectă în structura și forma finală.

Deși posedă numeroase facilități, am folosit doar posibilitatea de înlocuire a variabilelor din interiorul documentului cu variabilele reținute în cadrul cererii de concediu. Pentru înlocuire este necesar un HashMap ce păstrează drept cheie numele variabile codate în document, iar ca valoare, parametrul echivalent în cadrul cererii de concediu. Am păstrat într-o clasă statică metodele ce transformă obiectul de tip cerere de concediu în document word deoarece în cadrul imaginii de ansamblu ea nu are alt rol decât de transformer, apropiindu-se mai degrabă de programarea funcțională decât de programarea orientată pe obiect.

* + 1. Atribuțiile administratorului

Cum am menționat anterior există două roluri principale, cel de administrator și utilizatorul obișnuit. Deși administratorul nu poate depune cereri de concediu, el ca și angajat având două conturi, administratorul aprobă cererile de concediu, crează noi angajați și îi administrează prin intermediul interogării și vizualizării datelor acestora.

Administratorul are deplin acces la datele angajaților săi, și prin urmare la cererile efectuate de aceștia. Mai are posibilitatea să deschidă și să descarce atașamentele cererilor, să vadă numărul de zile cerute, zilele rămase în anul curent și trecut cât și alte detalii. Pentru a accepta sau respinge cererile în curs, administratorul trebuie să navigheze la pagina de ”Cereri”. Aici îi sunt listate toate cererile de la toți utilizatorii putând selecta câteva sau pe toate și le poate aproba/respinge. Aprobarea cererilor aduce cu sine modificarea unui câmp de tip boolean din interiorul bazei de date iar respingerea lor aduce cu sine ștergerea permanentă a cererii.

Pentru a vedea starea angajaților administratorul trebuie să intre în fereastra ”Listă angajați” unde aceștia vor fi afișați într-o structură de tabel, pentru a deschide detaliile unui angajat este suficient să dea click pe rândul pe care acesta este afișat și se deschide o nouă fereastră în care se listează toate cererile de concediu ale acestuia.

Adăugarea noilor angajați este tot o atribuție a administratorului, formularul cu datele necesare fiind în câmpul ”Adauga angajat” din navbar. Datele necesare sunt numele de utilizator,numele și prenumele persoanei, rolul contului, email-ul, ocupația și numărul de telefon. După completarea și submiterea formularului, se crează un obiect nou de tip User căruia i se asociază o parolă generată automat. Parola este compusă din 10 caractere aleatoare considerate sigure din punct de vedere al securității. După ce obiectul este salvat în baza de date se compune un email ce este trimis la adresa introdusă în interfață prin care se notifică utilizatorul de crearea contului. În acest email se regăsesc printre altele numele utilizatorului și parola, acesta fiind rugat ca la prima logare să își verifice și actualizeze contul cu o parolă nouă. Se dorește scoaterea self-enrollment-ului, prin acest mecansim putându-se controla identitatea noilor utilizatori, permițând crearea de noi conturi doar de persoane acreditate. Deși administratorul crează contul acesta nu are acces la parola utilizatorului, singura necesitate fiind adresa de email validă.

Pentru a evita aglomerarea cererilor zilelor de concediu administratorul are acces la o pagină specială de calendar în care sunt afișate toate cererile de concediu active pe luna curentă. Pentru crearea calendarului am folosit funcționalitatea Monthly.js de calendar. Pentru popularea calendarului este necesară o sursă de date sub format XML. Deși în ghidul de utilizare oferit de creatorul acestei biblioteci sursa de date este un fișier XML aflat pe server ce este adus prin apel de tip GET, am preferat să aduc datele în mod dinamic, după inspectarea codului din bibliotecă am observat că datele pot fi și sub format șir de caractere. Astfel încât am modificat în proprietățile obiectului de tip calendar astfel încât să se apeleze o metodă din controller ce întoarce un XML fără a mai persista vre-un fișier pe server. Structura datelor folosită de bibliotecă fiind diferită de structura cererilor de concediu din aplicație, a impus crearea unor obiecte noi și a unor translatori, construiți după modelul Interface Pattern. Folosind JAXB am transformat obiectele Java în structură arborescentă de tip XML ce este folosită mai departe de către interfață pentru construirea calendarului.

* + 1. Nivelul datelor

Pentru accesarea datelor am folosit Spring Data alături de Java Persistence API prin implementarea sa, Hibernate. Am dorit folosirea acestui tip de acces în detrimentul JDBC deoarece permite maparea automată a obiectelor pe tabele fără a mai fi necesară scrierea de cod SQL în interiorul șirurilor de caractere.

Dificultatea folosirii acestei modalități de accesare a datelor a constat în configurarea API-ului. Hibernate are o serie de funcționalități ce pot fi integrate cu Spring, printre care afișarea codului SQL în consolă pentru partea de debugging, stabilirea dialectului de SQL folosit, generarea de statistici și păstrarea cererilor în cache. Numele de utilizator și parola folosite pentru conectarea la baza de date se află în fișierul de configurare application.properties, pe care Spring le preia și le injectează în configurarea de Hibernate.

**public interface** UserHolidayPeriodsDao **extends** CrudRepository<UserHolidayPeriod, Integer> {  
 **public** ArrayList<UserHolidayPeriod> findByUserUserId(**int** userId);  
 **public** ArrayList<UserHolidayPeriod> findByUserUsernameAndIsApproved(String u,**boolean** app);  
 **public** ArrayList<UserHolidayPeriod> findByIsApproved(**boolean** isApproved);  
 **public** ArrayList<UserHolidayPeriod> findByIdIn(List<Integer> ids);  
 **public** UserHolidayPeriod findByIdAndUserUsername(**int** id,String username);  
}

Fig. 9 Interfață DAO folosind Spring Data

Un beneficiu major a fost ușurința creerii obiectelor de tip DAO și a metodelor de interogare a bazei de date. Spring Data permite crearea de noi repository-uri prin extinderea interfeței CrudRepository ce are doi parametrii, primul este clasa cu adnotarea Entity, entitatea ce mapează tabelul și al doilea este tipul cheii primare.

Pentru a crea diferite interogări este suficient să adăugăm o nouă declarare de metodă în această interfață, Spring ocupându-se de crearea efectivă a interogării. Forma interogării este dată de numele metodie, acesta fiind interpretat și transpus. Metodele trebuie să inceapă cu prefixul ”findBy”, urmat în formă camel case de celelalte reguli. Exemplu metoda ”findByIsApproved” va căuta în tabelul atașat entității UserHolidayPeriod doar acele obiecte ce au valoarea dată ca parametru. Pentru interogări mai complexe, folosind obiecte constituente ale entității deci folosind legătura cu alte tabele, în nume se adaugă numele obiectului constituent urmat de numele atributului obiectului. Spring folosește descrierea entității pentru stabilirea relației dintre obiecte și construiește cererea astfel încât să reflecte numele metodei. Compunerea de reguli se face folosind unul dintre identificatorii ”And”, ”Or”, ”greaterThan” și ”lesserThan”. Evitând scrierea de cod SQL în interiorul claselor Java se păstrează un context de programare, se evită greșelile de sintaxă, majoritatea IDE-urilor nu recunosc codul SQL scris în interiorul claselor și se păstreză forma de object oriented programming pur, nealterat.

* + 1. Baza de date

Pentru dezvoltarea acestei aplicații am folosit baza de date MySQL. Pentru această alegere am avut în minte câteva considerente : în primul rând am folosit o bază de date relațională datorită formei datelor, acestea având un grad redus de complexitate a informației. Informațiile simple, structurate în interfață sub formă de tabel duc ușor la o structură asemănătoare și în baza de date. Alt considerent a fost structura schemei bazei de date, cum obiectele manipulate în interiorul aplicației sunt relativ statice, neavând schimbări majore într-un interval de timp foarte scurt, o bază de date cu o schemă stabilită este mai eficientă decât o bază de date de tip NoSQL fără schemă. De asemenea permite dezvoltarea ulterioară de sisteme de analiză de productivitate. Datele păstrate in baze de date relaționale sunt definite de legi matematice precise ce permit într-o manieră elegantă introducerea lor în algoritmi de prelucrare și raportare.

Aplicația având dimensiuni reduse, fără facilități extrem de complexe a cerut o bază de date simplă, ușor de administrat. Spre deoserbire de baza de date Oracle, clientul MySQL are dimensiuni reduse, ocupând mai puține resurse pe sistemul în care este instalată. De asemenea facilitățile complexe precum controlul fin al accesului la baza de date, structura ierarhică compusă din baza de date, schemă, tabel complică procesul de dezvoltare și administrare deoarece aplicația nu va interacționa cu alte medii, baza de date nefiind partajată între mai multe aplicații. Astfel se formează imaginea bazei de date potrivite acestei aplicații, o bază de date relațională de dimensiuni reduse ce poate susține o aplicație ce nu va partaja datele cu alte aplicații, ce va manipula un număr relativ mic de date și ce va permite o eventuală analiză a acestora.

Pentru a respecta cu strictețe rigorile matematice impuse în domeniul bazelor de date relaționale, am adus-o la forma normala 1 (FN1) eliminând datele ce se repetă, pentru fiecare element nou introdus se generează automat o cheie primară iar tabelele se leagă între ele prin intermediul ”foreign key”.

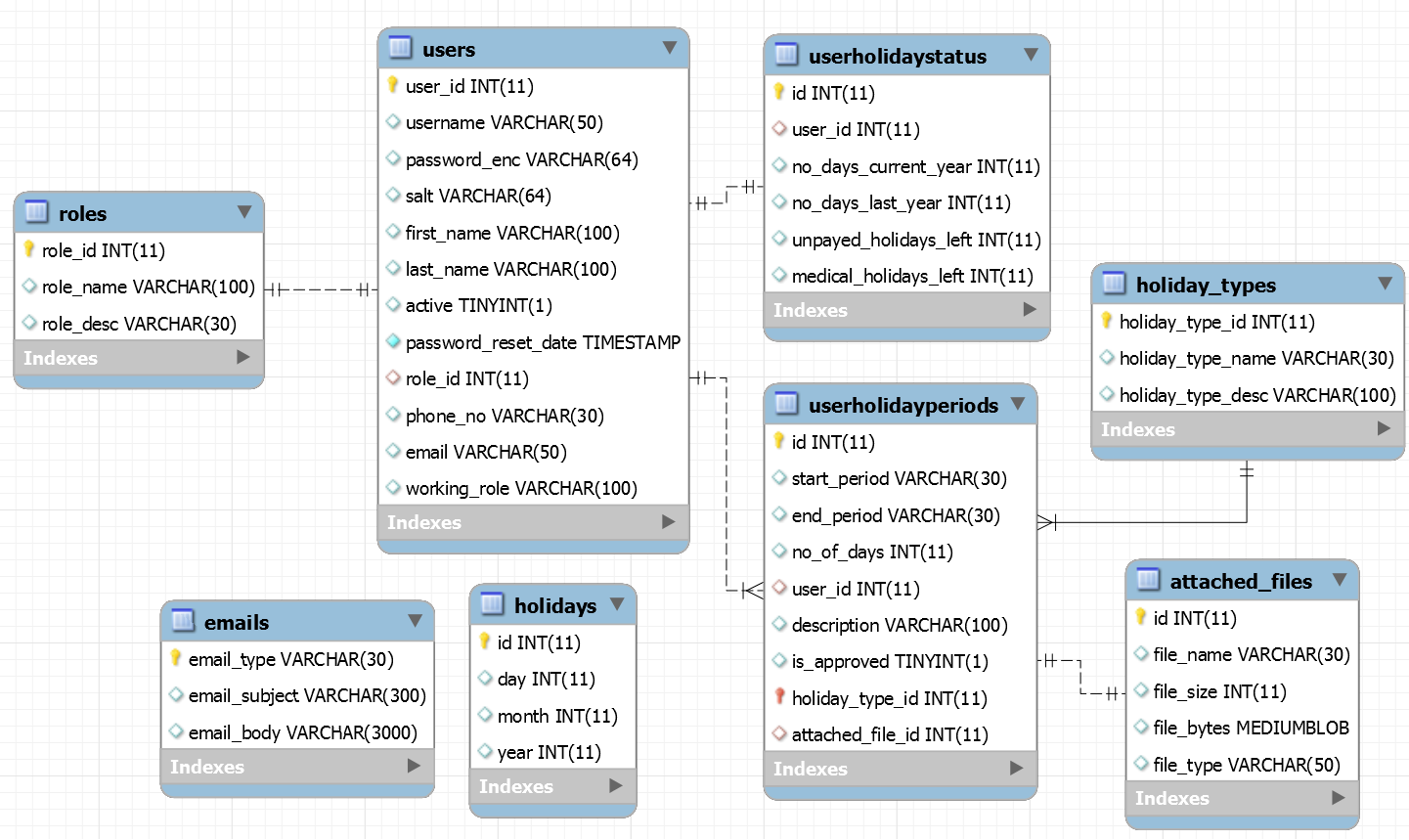


Fig. 11 Diagrama bazei de date

Din diagrama bazei de date se poate observa o structură generală a aplicației. De observat diferența dintre modelul de analiză descris în fig. 6 și modelul de proiectare redat prin fig. 11 deși păstrează unele elemente comune precum utilizatorul, detaliile despre situația cererilor de concediu în tabelul ”userholidaystatus” și cererile efectuate de utilizator în relația one-to-many în tabelul ”userholidayperiods” , pe parcursul dezvoltării s-au adăugat elemente noi precum tipuri de cereri de concediu, fișiere atașate, roluri la utilizatori, email-uri și zile libere declarate oficial.

Tabelul central din baza de date cât și elementul principal din aplicație este Utilizatori. Oferind servicii pentru angajați (ESS) la acest nivel se regăsesc cele mai multe date și se înteracționează cu cele mai multe tabele în mod direct cât și tranzitoriu. O eventuală refactorizare a aplicației va trebui să țină cont de dimensiunea și cantiatea de date persistată în acest tabel și de relațiile cu acesta. Pentru a mai prelua din complexitatea sa, am creat tabelele ”roles” și ”userholidaystatus” fiecare având în contextul său logic câte un rol, fiind legate în continuare de utilizator dar folosite în cadrul aplicației în alte contexte.

1. Performanța

Pentru a stabili performanța aplicației am folosit un tool special de testare ce permite verificarea încărcării, stresului și a ascalabilității aplicației numit LoadComplete[[43]](#footnote-33). Acesta permite urmărirea acțiunilor pe serverul de test și redarea acelor acțiuni folosind o serie de utilizatori virtuali. Testele au fost realizate pe mașina locală dar cu legătura la baza de date oferită de sistemul cloud oferit de Google pentru a simula o conexiune la distanță. Am ales această configurație deoarece consider că un element împortant în dezvoltarea oricărei aplicații de tip web este legătura la baza de date. O aplicație ce funcționează corect accesează cât se poate de rar baza de date, folosind cât mai puține conexiuni.



Scenariul de test ales acoperă funcționalitatea de bază a aplicației urmărind a trece prin principalii pași : autentificarea, încărcarea conținutului, trimiterea cererii de concediu și întoarcerea la pagina principală. Pentru a determina eficient caracteristicile aplicației, am stabilit mediul de testare prin folosirea a 12 utilizatori virtuali timp de 5 minute, începând cu un utilizator, adăugând pe rând odată la 15 secunde alt utilizator ce depunea cerereri. Criteriul principal de performanță la acest nivel este viteza de încărcare influențând pozitiv satisfacția utilizatorului. În fig. 19 se observă timpul relativ ridicat de încărcare pe cererea de autentificare. Acest timp se datorează trecerii prin două cereri succesive, prima de autentificare și validare a utilizatorului și a doua de încărcare a paginii principale. Ambele cereri necesită accesarea bazei de date pentru extragerea și validarea utilizatorului în baza de date astfel obținându-se timp de încărcare mari. Compunerea cererii presupune și crearea de noi fișiere prin intermediul template-ului, extragerea fișierului XML din fișierul .docx oferit ca template este un proces ce consumă timp. Alt motiv este aducerea resurselor de pe server, la încărcarea paginii de autentificare, se aduc de pe server fișierele JavaScript și CSS prelungind astfel perioada de timp necesară încărcării paginii.



În figura de mai sus se observă timpii de încărcare a paginii pe perioada a celor 5 minute de testare cu încărcarea celor 12 utilizatori. La minutul 3 și 45 de secunde se observă o anomalie, anomalie datorată clients1.google.com ce permite se ocupă de sugestiile de căutare și încărcarea datelor deja folosite. În medie se observă că încărcarea paginilor nu depășește 5 secunde oferind utilizatorilor un grad mediu de responsivitate a aplicației.

Pentru a ridica performanța aplicației se recomandă instalarea bazei de date pe același sistem în care este încărcată aplicația, folosirea unei baze de date remote deși aduce câteva avantaje precum posibilitatea de back-up și securitatea informațiilor, dezavantajul de overhead de comunicare o transformă într-o soluție ineficientă. De asemenea este recomandat ca cererile la baza de date să fie cât mai rare cu putință, iar resursele folosite de aplicație să fie aduse o singură dată la încărcarea aplicației, micșorând cantitatea de date și timpul de încărcare. De asemenea se recomandă folosirea ultimelor versiuni de browsere, Google Chrome, Mozilla Firefox și Microsoft Edge pentru a suporta toate facilitățile oferite de tehnologiile folosite în principal HTML5. O altă posibilitate de a crește performanța este modificarea generării documentelor de template și folisirea serverului local de SMTP pentru transmiterea de emailuri de la aplicație la utilizatori.

1. Concluzii

În aplicația de față am oferit posibilitea gestionării zilelor de concediu prin intermediul unei aplicații web. Am permis utilizatorilor să creeze cereri noi de concediu ce sunt conforme legilor de acordare a acestora pe teritoriul României, acoperind cele 3 tipuri generice de concediu : concediu de odihnă, fără plată și medical, cel din urmă având necesități suplimentare precum încărcarea fișelor medicale sau a fișelor de externare. Utilizatorii pot prin intermediul aplicației să vadă numărul de zile de concediu rămase pentru fiecare tip în parte, să își administreze datele personale și să vadă un istoric al cererilor aprobate.

Administratorii au posibilitatea să adauge utilizatori noi, păstrând confidențialitatea datelor personale față de noii utilizatori, să observe situația angajaților săi la nivel de zile de concediu, să interogheze istoricul acestora și să respingă sau aprobe noile cereri. Pentru a controla și mai eficient, administratorul are acces la un calendar în care sunt afișate zilele de concediu aprobate din perioada următoare acesta putând pe baza lui să ia decizii de a aproba eventualele cereri.

Notificarea utilizatorilor pentru fiecare dintre acțiunile de depunere de cerere sau aprobare se face prin intermediul emailului transmis automat de către aplicație. Prin această metodă se facilitează comunicarea asincronă între cele două entități, ne fiind necesar ca nici una dintre părți să folosească aplicația mai mult decât este necesar.

Am dezvoltat aplicația având în considerare acesarea cât mai eficientă a datelor, o comunicare cât mai redusă și cât mai robustă între entitățile ce formează ansamblul aplicației iar arhitectura ei permite integrarea de noi module și facilitează dezvoltarea ulterioară.

În viitor aș dori să rezolv cerința clientului de integrare a aplicației cu Slack, aplicație ce permite transmiterea de mesaje între diferiți utilizatori, permițând acestora distribuirea de cunoștințe (knowledge sharing) prin link-uri sau fișiere ce pot fi apoi vizualizate în cadrul aplicației web.

1. ”ESA Annual Report 2015”, http://www.theesa.com/wp-content/uploads/2016/04/ESA-Annual-Report-2015-1.pdf [↑](#footnote-ref-1)
2. ”Internet Users in the World”, http://www.internetlivestats.com/internet-users/ [↑](#footnote-ref-2)
3. http://www.internetsociety.org/internet/what-internet/history-internet/brief-history-internet [↑](#footnote-ref-3)
4. Bibliografie

   ” MVC pattern overview”, https://developer.chrome.com/apps/app\_frameworks [↑](#endnote-ref-1)
5. "What’s is a web app? Here’s Our Definition", 2011, http://web.appstorm.net/general/opinion/what-is-a-web-app-heres-our-definition/ [↑](#endnote-ref-2)
6. http://singlepageappbook.com/goal.html [↑](#footnote-ref-4)
7. ”What's a Page?”, https://johnpapa.net/pageinspa/ [↑](#footnote-ref-5)
8. ”The disadvantages of Single Page Applications”, http://adamsilver.io/articles/the-disadvantages-of-single-page-applications/ [↑](#footnote-ref-6)
9. "Ajax : A new Approach to Web Applications" , Garrett, James, 2005, https://web.archive.org/web/20080702075113/http://www.adaptivepath.com/ideas/essays/archives/000385.php [↑](#endnote-ref-3)
10. "jQuery Api", http://api.jquery.com/ [↑](#endnote-ref-4)
11. ” https://api.jquery.com/category/selectors”, https://api.jquery.com/category/selectors/ [↑](#footnote-ref-7)
12. ”Grid system”, http://v4-alpha.getbootstrap.com/layout/grid/ [↑](#footnote-ref-8)
13. ”Advanced HTML, cursul 3 Programare Web, Conf.dr.ing Ciprian Dobre” [↑](#footnote-ref-9)
14. ” Ogg Theora, H.264 and the HTML 5 Browser Squabble”, http://www.roughlydrafted.com/2009/07/06/ogg-theora-h-264-and-the-html-5-browser-squabble/ [↑](#footnote-ref-10)
15. ”Status of This Document”, https://dev.w3.org/html5/webdatabase/ [↑](#footnote-ref-11)
16. http://www.thymeleaf.org/index.html [↑](#endnote-ref-5)
17. ”Why Use the Spring Framework?”, http://www.wrox.com/WileyCDA/Section/Why-Use-the-Spring-Framework-.id-130098.html [↑](#footnote-ref-12)
18. ” The DispatcherServlet”, <http://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm> [↑](#footnote-ref-13)
19. http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html [↑](#endnote-ref-6)
20. ”Registration of multipart/form-data”, https://tools.ietf.org/html/rfc1867#section-7 [↑](#footnote-ref-14)
21. http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/multipart/MultipartFile.html [↑](#footnote-ref-15)
22. ”Uploading Files”, https://spring.io/guides/gs/uploading-files/ [↑](#footnote-ref-16)
23. ”Views and resolving them”, http://docs.spring.io/autorepo/docs/spring-framework/3.0.0.M3/reference/html/ch16s05.html [↑](#footnote-ref-17)
24. <http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/> [↑](#footnote-ref-18)
25. ”Why we need Spring Boot”, <http://javabeat.net/spring-boot/> [↑](#footnote-ref-19)
26. <http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/>, Running your applicaiton, Hot swapping, SpringApplication, Web environment [↑](#endnote-ref-7)
27. [↑](#footnote-ref-20)
28. [↑](#footnote-ref-21)
29. http://docs.spring.io/spring-security/site/docs/3.0.x/reference/introduction.html [↑](#endnote-ref-8)
30. http://docs.spring.io/spring-security/site/docs/3.0.x/reference/technical-overview.html [↑](#endnote-ref-9)
31. ”Filter Ordering”, <http://docs.spring.io/spring-security/site/docs/3.0.x/reference/security-filter-chain.html> [↑](#footnote-ref-22)
32. ”Web Securiy Expressions”, <http://docs.spring.io/spring-security/site/docs/3.0.x/reference/el-access.html> [↑](#footnote-ref-23)
33. https://tools.ietf.org/html/rfc7519 [↑](#endnote-ref-10)
34. https://docs.jboss.org/hibernate/orm/5.1/userguide/html\_single/images/architecture/data\_access\_layers.svg [↑](#footnote-ref-24)
35. http://exceltemplates.net/calendar/excel-employee-vacation-tracking-calendar-template/ [↑](#footnote-ref-25)
36. https://timetastic.co.uk/ [↑](#footnote-ref-26)
37. https://www.whosoff.com [↑](#footnote-ref-27)
38. https://www.whosoff.com/tour/ [↑](#footnote-ref-28)
39. https://f2.washington.edu/fm/payroll/payroll/ESS [↑](#footnote-ref-29)
40. https://idp2.shib.umn.edu/idp/umn/login [↑](#footnote-ref-30)
41. https://ess.deere.com [↑](#footnote-ref-31)
42. http://kthornbloom.com/monthly/ [↑](#footnote-ref-32)
43. https://smartbear.com/product/loadcomplete/overview/ [↑](#footnote-ref-33)